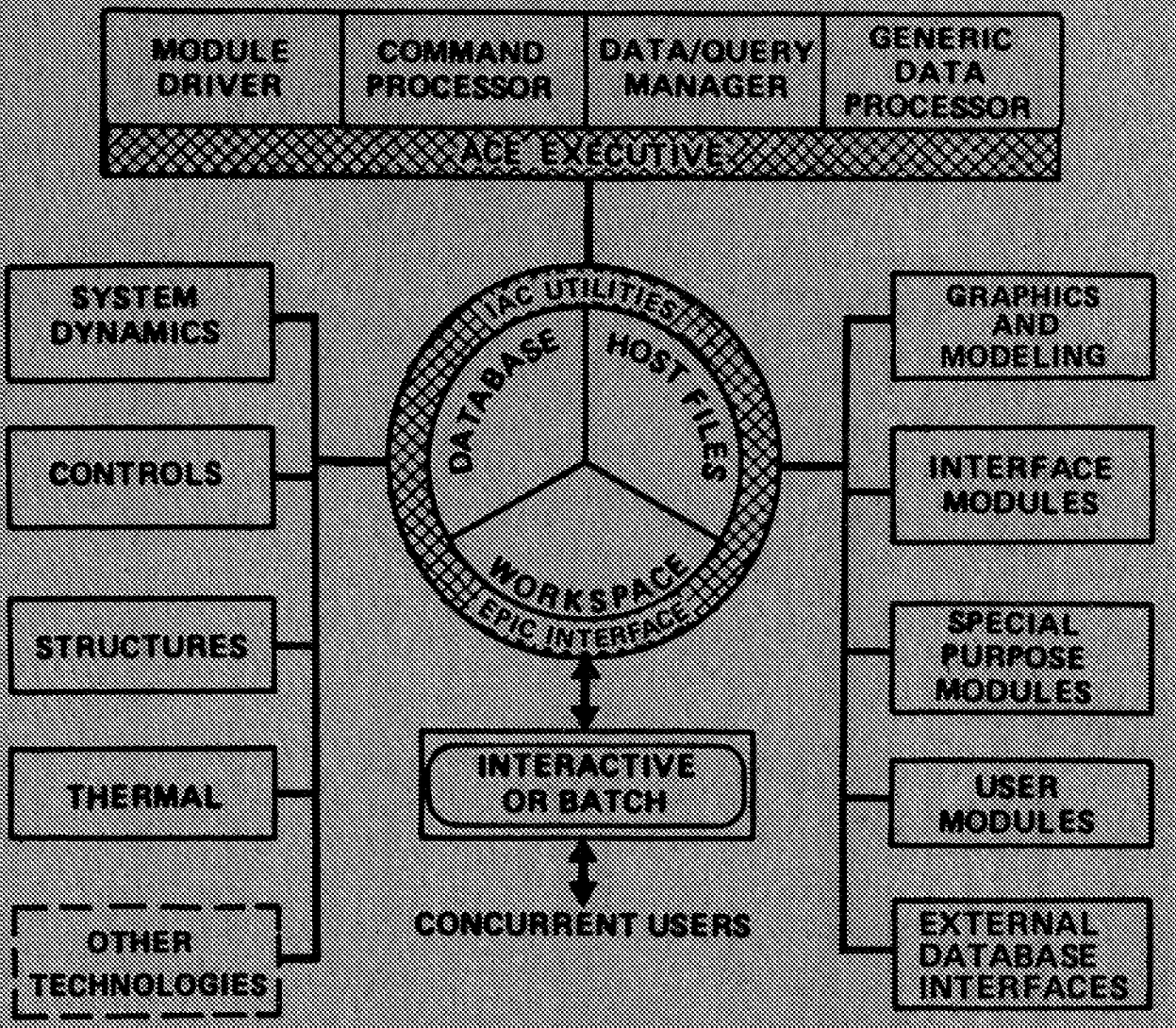
NASA-CR-175300-VOL-3

Boeing/NASA/COSMIC Extended Documentation
(Based Upon NASA Contractor Report 175300
and Contract NAS5-25767

DO NOT DESTROY
RETURN TO LIBRARY
DEPT. 022

Integrated Analysis Capability (IAC)

User Manual (Level 1.5)



LRN 8883
LM104602E

NASA
National Aeronautics and
Space Administration

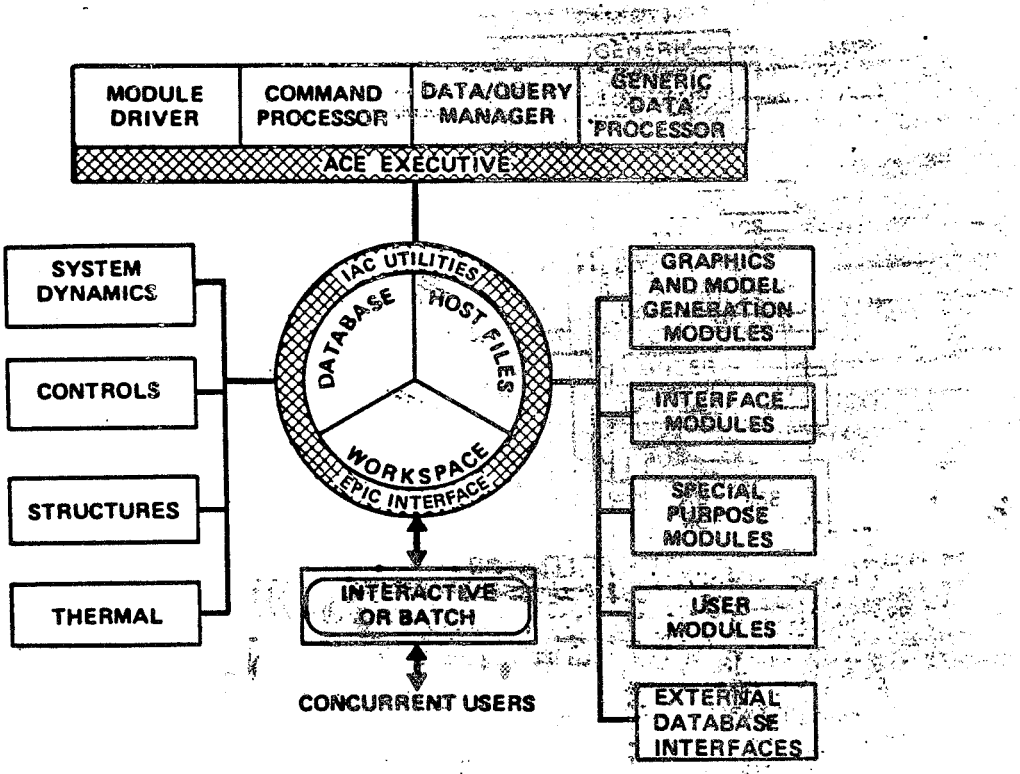
May 1985

LM104602E

COSMIC PROGRAM # GSC-12994
Volume 3 and 4

NASA Contractor Report 175196

IAC Executive Summary



William J. Walker, Robert G. Vos,
Gerald A. Price and Elliott W. Brogren
Boeing Aerospace Company
Seattle, Washington

Prepared for
Goddard Space Flight Center
Greenbelt, Maryland
under Contract NAS5-25767

National Aeronautics and
Space Administration

1984

Table of Contents

		Page
	List of Figures	ii
	Summary	iii
1.0	Introduction	1
2.0	Integrated Analysis Architecture	5
3.0	Analysis Capabilities	8
	3.1 Thermal Analysis	12
	3.2 Structural Analysis	15
	3.3 Control Analysis	17
	3.4 Analysis Enhancements	21
4.0	Executive and Data Management	26
5.0	Conclusions and Recommendations	34
6.0	References	37

List of Figures

		Page
1-1	Typical IAC Utilization	38
1-2	IAC Data Flow Schematic	39
2-1	Two IAC Products	40
2-2	IAC Architecture Summary	40
3-1	IAC Standard Solution Paths	41
3-2	IAC Standard Application Modules	42
3-3	ORACLS Standard Driver REGEST	43
3-4	IAC Control Data Flow	43
3-5	IAC Data Flow Techniques	44
4-1	IAC Task Control Schematic	45
4-2	IAC Database Schematic	46
4-3	IAC Workspace Schematic	46
4-4	IAC Data Structures	47

SUMMARY

This executive summary presents an overview of the Level 1 Integrated Analysis Capability (IAC). IAC combines operational modules from the disciplines of structures, controls and thermodynamics with an executive and data management system to yield a highly effective multi-disciplinary code. Special attention has been given to (1) user requirements such as data handling and on-line assistance with operational features, and (2) the ability to add specific modules of the user's own choice to the system. IAC has evolved through three phases over a five year development period and has provided extensive usage experience at the Boeing Aerospace Company and at the Goddard Space Flight Center. Users interested in operational details of the IAC are referred to the Level 1 User Manual. Finally recommendations are presented for continued expansion of IAC.

1.0 INTRODUCTION

The objective of the Integrated Analysis Capability (IAC) has been to provide a highly effective, interactive analysis tool for integrated design of large space structures. IAC has focused on the technical disciplines of structures (both statics and dynamics), controls and thermodynamics. These disciplines are represented by state-of-the-art modules which existed at the start of the IAC development and are continuing to be maintained either by government laboratories or by private industry.

The IAC development started in June of 1979 and is scheduled for completion in July 1984. During this five year period, the IAC software has evolved through three phases using funding of the NASA-Goddard Space Flight Center (GSFC). Phase I was a proof of concept code developed during a ten month study which established the IAC development plan (Ref. 1). Phase II saw the first operational IAC called "Level 0", with an associated user manual, delivered to GSFC. A functional specification (Ref. 2) for the IAC was also written during this time.

IAC Level 0, while focusing on the thermal/structural interface, also provided an overall multi-discipline integrated capability. Level 0 has been used extensively at the Boeing Aerospace Company (BAC). Figure 1-1 shows a typical graph of engineering usage of this code on one of five VAX machines on which it is available within BAC. It has been found that the IAC data query and plotting features facilitate productivity by relieving many of the drudgeries of analysis and data reduction.

In addition to its technology interfacing and data management capabilities, IAC has had a significant impact as a manager of technical software modules. Software configuration control and accessibility have been enhanced, and the user interface for these modules has been improved and made more uniform. For example, IAC provides standard handling of all module run-time parameters, associated on-line help and example information, and the ability to easily link user-defined capabilities into many modules while maintaining the system defined standard software. The result is that the modules incorpo-

rated within IAC have become more productive and more accessible to a wider group of users, and the user learning curve for each individual module has become less costly.

In terms of integrated analysis, IAC Level 0 has been used on a number of technically challenging problems at BAC.

Avionics Boxes:	Design of the boxes required both thermal and vibration analysis.
Rocket nozzles:	Failure predictions of several nozzles required both thermal and static analysis.
Toroidal Tank Design:	Parametric analyses of tank cross section and wall thickness were conducted for strength and modal characteristics.
Printed circuit boards:	Thermal/elastic/plastic/creep analysis of integrated circuit chip carriers and solder joints; study of geometric parameters and fatigue life.
Space Station Conceptual Design:	Obtained dynamic characteristics and conducted control systems analysis to study system design parameters.

The experience gained and cost savings obtained by using IAC Level 0 have demonstrated the advantages of this code. Not all engineering problems are truly integrated but most engineering problems have an interdependence of data. IAC has shown itself to be an effective assistance in automating data transfer from one analysis to another.

Phase III has extended the Level 0 code with significant enhancements to the executive/user interface and the general data management capabilities, extension of the database cataloging mechanism to include both structured and unstructured files, incorporation of several additional controls modules, and an interface with the database module RIM (Relational Information Management). The extended IAC system is now referred to as "Level 1". Level 1

capabilities are summarized in the sections below and user operational instructions are presented in the revised User Manual (Ref. 6).

IAC is built around the data flow required to solve integrated problems. Figure 1-2 shows the IAC data-flow schematic which begins with various system math models. Section 3 below discusses the IAC analysis capability where a different interpretation is put on Figure 1-2. There data flow is discussed in terms of solution paths (technical data flow requirements), thus treating the IAC features one at a time.

The architecture of the IAC system is shown on the cover of this summary by the IAC logo. We feel the logo shows better than words can explain how the IAC components fit together. A significant piece of this code is the executive and data management software, discussed in Section 4. Two ideas have been foremost in the development of this area of the IAC system. First, keep the user in mind. The executive has tried to be as user friendly as possible, and provides extensive on-line assistance in the way of explanatory help and examples for the user. Second, the executive and system software have provided for modular additions to the code. Here at BAC we have expanded the technical capability of IAC by incorporating several additional modules under IR&D funding. Every organization will have some unique modules which its users will want to interface with IAC, and the system has been developed to facilitate this kind of growth.

Naturally no computer code has all the features you want on its first release. Our original development plan has not been fully carried out due to funding constraints. We do feel there is a priority set of requirements for improving the Level 1 IAC operational features. We have provided this set of recommendations for further IAC development.

The subsequent section provides you with an overview of the IAC. We hope it will cause you to say "that system can help me, I'll try it", for we feel it can.

IAC is the result of concentrated and dedicated effort on the part of many people from BAC, the Boeing Computer Services (BCS), and GSFC. These people are to be congratulated for the success of their effort. From the Goddard Space Flight Center, Joseph P. Young was Program Director; he was assisted by Harold P. Frisch, Gary K. Jones, and Joseph T. Skladany. From Boeing, William J. Walker was Program Manager, Dr. Robert G. Vos was Technical Director, and David L. Beste was Director of Computing Operations; Elliott W. Brogren, Craig C. Chun, Richard M. Gates, Joseph W. Gregg, Lauri H. Hillberg, Dr. Robert M. Jones, Gerald A. Price, Suzanne M. Williams, and Sherry A. Winkleblack contributed to the effort.

2.0 INTEGRATED ANALYSIS ARCHITECTURE

IAC Products—The IAC system architecture can be viewed as providing two products:

1. A specific inter-disciplinary analysis capability having a set of interfaced technical modules that deal with, in the case of current Level 1, the coupling of thermal, structures and controls disciplines.
2. A general framework product which serves as an "integrating base" whereby user groups can add desired modules and disciplines.

These two products are shown schematically in Figure 2-1. The diagram at the left of the figure exemplifies a specific capability, i.e., the current system of IAC thermal, structures and controls analysis modules. The diagram at the right depicts the general integrating framework, into which "X-Y-Z" brand modules of a particular user group can be incorporated. Note that in either case the core software is identical.

Overview of Architecture—The IAC architecture is summarized by Figure 2-2. The ACE executive module, shown at the top of the figure, relies upon the same set of standard IAC utilities which are documented and available for use by all modules in the system. The architecture of the ACE (Analysis Capability Executive) is highly integrated, in the sense that it provides several major functions within a single executable module. ACE contains a general driver code to execute other application modules; an input command processor which handles language syntax, prompting, help information, etc.; an extensive data/query management capability which allows the user to create, manipulate, and evaluate various types of data; and a generic data processor which provides several useful capabilities, including an interface between IAC and "foreign" module character-formatted data. In contrast to its highly integrated internal architecture, the executive is independent from any requirements of a particular application module, thus facilitating the substitution or addition of new modules into the system.

The IAC concept of overall data organization and storage is summarized by the three data areas shown in the center of Figure 2-2. Design and use of the workspace, database and host file areas are discussed in Section 4. These areas collectively store all IAC data, and support required computations and query.

Module interfacing is facilitated by the EPIC (Enhanced Program Interface Capability) software, described in Section 3.4. The modules are generally executed via the executive program, with user specified options for run parameters and IAC/module data flow. However, any IAC module may also be executed directly by a user, or from within another IAC module. The groups of technical modules shown on the left side of Figure 2-2 depict the current specific analysis capability. Module groups displayed on the right side are associated with functions such as graphical data display, model building and viewing, inter-module data flow, model transformation and special data handling, and management support for user owned software. The external database interfaces block is discussed in Section 4.0.

Multiple concurrent users may interface with IAC to perform both interactive and batch tasks.

Architecture Design—Design of the IAC architecture has been driven by several considerations associated with the specific nature and objectives of IAC. IAC is an interactive but in-depth computational analysis package, applicable to the full range of activity from conceptual through final design. There is a need to support a variety of different analysts and user groups, including both technical specialists as well as multi-disciplinarian types of engineering personnel, and involving diverse technical modules and analysis techniques. In addition, the IAC system is designed to have a long life potential, by emphasizing a highly modular construction and by catering to modern hardware and software concepts. These considerations have led to the following overall IAC design characteristics.

1. A modular engineer-in-the-loop mode of operation is accommodated in preference to the alternative of a more monolithic and highly automated

process. That is, the various capabilities are provided as independent logical building blocks which are selected and combined as required by the analyst for a specific application.

2. The IAC system/module interface is designed to support a simple "plug-in" implementation for new modules. This facilitates user maintenance and incorporation of additional capabilities.
3. Existing technical modules are used whenever feasible. This maintains compatibility with existing user groups and provides access to sophisticated and in-depth technical modules necessary for final design type activities.
4. A primary emphasis is placed upon computational requirements and the needs of the technical applications programmer. The system supports efficient management of module defined data, but also provides for a variety of IAC defined data structures and for detailed data manipulation and query.
5. IAC is designed to take advantage of recent advances in modern computing hardware and software. Modern host operating system techniques are used to facilitate concurrent data access and effective task control transfer, and it is assumed that generous amounts of real and/or virtual RAM memory are available. FORTRAN 77 is the primary IAC programming language, but is supplemented where necessary by small amounts of host dependent code. The initial IAC development has been accomplished on a DEC VAX computer.

3.0 ANALYSIS CAPABILITIES

Solution Paths—The term "solution path" refers to a standard or user defined sequence of IAC provided operations. A solution path is often designed to handle a class of problems, which consists of a number of selectable options and variations, rather than a rigidly predefined and automated process. An engineer-in-the-loop mode of operation is therefore possible and encouraged.

Figure 3-1 illustrates five standard solution paths which have been defined within the thermal/structures/controls specific capability. The solid lines of Paths I to IV indicate capabilities which have been implemented. The standalone (uncoupled) operation of each technology or major technical module is defined to be Solution Path I. Paths II through V involve an increasing degree of inter-disciplinary coupling and correspondingly greater complexity.

Solution Path II provides thermal deformations, for either a steady-state or transient case, via the coupling of a thermal analyzer with a structural statics analyzer. A modeling transformation is provided by IAC to handle data flow between the generally incompatible thermal and structural models.

Path III accomplishes a structural/control analysis, in either the frequency or time domain, by transferring required data from a structural normal modes analysis to a system dynamics module. The linearized plant model can be generated and saved for use by a control system design module.

The thermal/structural/control Solution Path IV provides a time-domain structural-control analysis, as in Path III, but with the added capability to handle time varying thermal loading effects. The thermal effects are assumed to be weakly coupled with the system dynamics behavior, i.e., the thermal loads can be computed a priori without direct regard for the dynamic changes in configuration. Although the primary purpose of Solution Path IV is to handle motion independent thermal loadings, the requirements for handling similar type mechanical loadings are virtually identical, and both types of loading may be included independently or in a combined fashion.

Finally, the Solution Path V is planned at some future time to provide a fully coupled analysis in the frequency domain and is directed at problems such as thermal flutter of long spacecraft members.

Modules—In addition to the executive module, there are several other types of modules available within the IAC system. These modules are conveniently listed within five different classes:

1. Technical modules (e.g., NASTRAN and DISCOS) which provide standalone computational capability as well as support for various solution paths;
2. Graphics and model generation (model building and viewing) modules;
3. Interface modules which provide required data flow between IAC and other modules;
4. Special purpose modules which provide miscellaneous important support functions; and
5. User modules.

The first four classes of modules are directly incorporated into the IAC system. This means that IAC associates a predefined name and set of Job Control Language with the module, a table of module run parameters and characteristics has been defined, and certain IAC/module data flow interfaces may have been developed. Such modules are generally accessible to all users at the particular IAC installation.

The user modules in the last class consist of arbitrary executable programs or Job Control Language procedures, which have no predefined relationship to the IAC system. A user module may be executed from within the IAC executive in much the same manner in which other modules are executed. The user module concept is valuable for handling infrequently used or user-specific software, in testing of new software, and in developing ad hoc modules and solution paths.

Interface modules are discussed in Section 3. The current set of standard technical, graphics, and special purpose modules available to IAC users are listed in Figure 3-2. It is anticipated that this list will be extended by

each particular user group, as user perferred modules are substituted and company specific proprietary modules are added. Such has been the case at Boeing, with the incorporation of several additional modules in the thermal, structures and controls disciplines. As noted earlier, a primary objective of the IAC architecture design is to permit simple incorporation of the modules preferred by a particular user group. The four technical module groups - thermal, structural, controls, and system dynamics - are discussed below.

The selected thermal and structural modules are generally well known within the technical community. It will be readily apparent to those familiar with the designated thermal and structural modules that there is some duplication of capability, e.g., SINDA/NASTRAN. This is due in part to a decision that both finite difference and finite element thermal codes should be available. More importantly, it is the result of a conscious effort to provide alternate technical modules within the specific IAC product in order to support as wide an existing user community as practical.

In the controls group, ORACLS (Optimal Regulator Algorithms for the Control of Linear Systems) is a collection of subroutines for the design of regulators and associated optimal filters, for systems described by continuous differential or discrete difference equations. The library of computer programs and subroutines SAMSAN (SAMpled System ANalysis) contains a set of algorithms for design and evaluation of large order control systems, with emphasis on sampled system analysis. MODEL (Multi Optional Differential Equation Language) supports both time domain controls and control/structure interaction analysis, and translates a differential or difference equation input language to technically equivalent FORTRAN programs.

The system dynamics module DISCOS (Dynamic Interaction Simulation of COntrols and Structures) is a spacecraft oriented simulation program that takes into account all gyrodynamic interaction effects between an active control system and a plant definable as a system of two or more flexible and/or rigid bodies; it can be used for either nonlinear time domain analysis or linear frequency domain analysis.

Referring to the graphics class of modules, PLOT and MOSAIC were specifically designed to display data structures from an IAC database, and are based upon the DI-3000 Core standard graphics support package. PLOT generates vector displays of tabular data in the form of curves, charts, correlation tables, etc., while MOSAIC generates color raster displays of either tabular or array type data. Major proprietary model building and viewing systems such as MOVIEBYU, SUPERTAB and PATRAN are in the process of being interfaced with the IAC database. It should be noted that the set of preferred model building and viewing systems is expected to vary considerably from one IAC user group to another.

The special purpose class of IAC modules provides several diverse functions. MIMIC (Model Integration via Mesh Interpolation Coefficients) is a module developed to provide data flow between different IAC math models, e.g., thermal and structural; it also provides a general capability for interpolation of array type data (based on spatial coordinates, time values, etc. The RIM (Relational Information Management) database system is a module which has been incorporated within IAC as an additional standalone query processor; IAC/RIM data flow has been provided for relational type data structures, and it is anticipated that RIM will serve as a prototype model for interfacing between IAC and other data management systems. LINK is used to simplify the incorporation of user subroutines into IAC modules, thus supporting the creation of user specific module versions; it automatically manages the module standard software and any associated support libraries, and links these with the user supplied subroutines to create a new executable module.

3.1 THERMAL ANALYSIS

The IAC basic thermal analysis capability is embodied in three main modules—NASTRAN, SINDA, and TRASYS II. NASTRAN and SINDA give the user a choice of finite element or finite difference (nodal network) techniques, respectively, for steady-state or transient thermal analysis. These modules may be run in a stand-alone mode or in support of other solution paths, i.e., to furnish temperature data for structural or structural/control solutions. The TRASYS II module provides radiation view factors, radiation exchange data, and space environment natural thermal radiation data for input to either of the primary thermal analysis modules. Some degree of redundancy exists between the capabilities of TRASYS II and the built-in radiation subroutines in SINDA and, for users with access to MSC NASTRAN, the VIEW module within that program.

NASTRAN Thermal Analyzer—Two special NASTRAN DMAP sequences are provided within IAC, for execution of the NASTRAN Thermal Analyzer (NTA) in the nonlinear steady-state and transient thermal modes. These sequences provide for node/time/temperature data and similar heat flux and temperature gradient data to be written to a special output file. This output file can later be read by the IAC interface module INTA, to insert appropriate data into the database. All standard output data normally available via the standard NASTRAN case control output requests are also permitted. Requests for plotted output, via the standard NASTRAN data identification and format controls, will result in the generation of an output plot data file, suitable for processing by the NASTRAN graphics module, TECKPLOT.

Depending upon user preferences and problem requirements, the NTA thermal finite element model may be identical to a NASTRAN structural model to be analyzed, or may differ in part or totally. In the case of identical models IAC provides for direct transfer of nodal temperature output from NTA to the structural analysis. For models that differ in geometric definition, the MIMIC interpolation module (see Section 3.4) may be used to map the NTA output temperatures onto the structural model.

SINDA—The IAC SINDA module incorporates all the capabilities of current versions of SINDA, including the SINFLO fluid flow package and all standard SINDA auxiliary and utility subroutines. Provision for insertion of user-supplied FORTRAN-like statements and user-supplied subroutines is retained. The implementation of SINDA within IAC avoids modification of the SINDA code and yet provides a number of options for output of data to the database plus a general capability for adding further data flow options in the future.

No special requirements exist for coding input data to run SINDA within IAC. All controls and options are selected as part of the run command. In addition to the standard SINDA outputs, obtained in the usual way via calls to the appropriate built-in subroutines and written in the host directory, the IAC SINDA generates a special file of output data for later storage in the database. Included in this file are nodal temperatures, capacitances, conductances, applied heat loads, conductor heat flows, and nodal net heat flows. A parameter is specified in the run command which, for transient analyses, provides for output of these data at a different time interval from that specified for the standard SINDA output calls, if desired. The special output file may be processed by the module ISIN to store output data selected by the user in the database. These data may then be easily formatted for plotting, printing, or input to a structural analytical model or to the MIMIC interpolation module.

TRASYS II—The TRASYS II module implemented within IAC incorporates all the capabilities and options available in the standard TRASYS II program. An option was added, activated by a subroutine call specified in the input file by the user, to format output radiation exchange factors for direct input to NTA or SINDA. Since the most common use of TRASYS is to furnish data for direct use in these thermal analyzers, no special files are generated for direct processing of output data for storage in the database. Much of the standard output data written to host files, however, may be transformed into the database by means of the IAC data handling and query operations.

One of the standard output files TRASYS writes to the host directory is a plot file. This plot file may be processed by the graphics module, TRASPLOT,

which is part of the IAC TRASYS II module, for displays of model geometry, flux vector representations, and view and shadow relationships. Two-dimensional graphs of data, such as heat flux versus time, may be obtained from the output host files and the database query and plotting operations.

3.2 STRUCTURAL ANALYSIS

The current IAC structural analysis capabilities depend largely on the MSC NASTRAN module. COMSIC NASTRAN has also been incorporated and supports standalone analysis as well as certain structural/control data flow.

Execution of MSC NASTRAN within IAC provides access to all of the MSC supplied NASTRAN structural analysis capabilities, plus additional IAC supplied capabilities. The associated TECKPLOT module is provided for graphical display of NASTRAN computed results. IAC provides enhanced NASTRAN structural capabilities for normal modes and for static deformation analysis.

Normal-Modes—A special NASTRAN DMAP sequence is provided which allows several types of normal-modes analysis data, including nodal locations, nodal mass data, mode shapes, and modal stiffness and damping matrices, to be computed and saved. This data can then be processed by the IAC interface module INDA and stored in an IAC database, for later query and graphical display or for transfer to other modules. A primary use of this data is for input to the DISCOS system dynamics module. (See the structural/control and thermal/structural/control solution paths III and IV, described in Section 3.0.)

Static Deformation—A special NASTRAN DMAP sequence is provided which allows several types of static analysis data, including transient nodal and modal displacements, and various element quantities such as stresses and strains, energies and nodal-force-balances, to be computed and saved. Several IAC interface modules are provided to perform associated pre- and post-processing tasks. The INSAT module allows nodal temperature data, stored in an IAC database, to be used in generating NASTRAN thermal load sets; the load sets are automatically inserted into a NASTRAN input file. The INSAM module accomplishes a nodal-to-modal transformation of the NASTRAN statics model; again, the required definitions are automatically inserted into the NASTRAN input file. This nodal-to-modal transformation process can be utilized for various general purposes; it has particular application to the thermal/structural/control solution path IV, in which the thermal analysis is performed on a nodal basis, but the system dynamics analysis requires timewise

varying thermal deformations on a modal basis. Finally, the INSA module can process the NASTRAN saved static analysis data and store it in an IAC database, for later query and graphical display or for transfer to other modules.

3.3 CONTROL ANALYSIS

The IAC control system analysis tools encompass a wide range of control system analysis and synthesis techniques coupled with IAC interfacing software to analyze control interaction with structures and thermal effects. Control modules available within IAC are: 1) ORACLS subroutines for analyzing and synthesizing control systems utilizing modern optimal control techniques based on Linear-Quadratic-Gaussian (LQG) theory; 2) SAMSAN subroutines for analyzing control systems using classical techniques with code optimized for large order dynamic systems; 3) the MODEL program to simulate and plot time responses of linear and nonlinear control systems; and 4) DISCOS for time domain and frequency domain analysis of multiple connected flexible bodies, subject to a user defined control system and possible thermally induced deformations. These modules, and the major data flow between them, are discussed in the following topics.

ORACLS—ORACLS is a self contained collection of subroutines for analysis and synthesis of controllers and estimators, emphasizing modern optimal control methods. ORACLS programs transfer arrays to and from an IAC database via IAC provided module-callable utility routines. The ORACLS program may be executed within IAC in two modes: (1) user defined programs, and (2) standard IAC provided programs.

User defined programs can be developed by the designer to implement a wide variety of design approaches based on LQG techniques. IAC data storage and manipulation capabilities coupled to the ORACLS program provide a unique capability for rapid development of user friendly design tools. At Boeing several user defined programs have been developed to support IR&D programs with good success.

Two standard ORACLS driver programs are provided within IAC. REGEST solves the full state feedback linear optimal regulator/estimator/compensator problem for either continuous or discrete systems. TIMFST is a computationally efficient program to compute and plot time responses of closed loop systems defined by IAC stored matrices.

REGEST is an interactive program that performs iterative solutions for the linear optimal regulator, estimator, and/or compensator problem (see Figure 3-3). A user defined bulk input file allows selection of regulator and estimator options, selection of continuous and discrete systems, and selection of various print and debug options. Interactive prompts are given to allow the user to select output to be printed or stored, change design weighting matrices, select alpha shifts or create a plot file which can be accessed by the IAC PLOT module. REGEST architecture provides the capability to access the IAC executive and data query capability during an interactive session. This capability significantly enhances data manipulation and program efficiency. Intermediate results can be stored in the database to be accessed later for continued design synthesis.

TIMFST is an interactive program, which computes and plots time responses for open and closed loop systems. A user defined input file selects IAC input arrays to be extracted from the database. The program computes the user selected controller configuration system matrices and transforms them into modal coordinates. This results in sparse matrices allowing item by item response computation for efficient computer operation. Interactive prompts allow the user to select the type of control system to be analyzed and which states are to be printed and plotted. Debug options are also provided.

SAMSAN—SAMSAN is an expandable library of state-of-the-art algorithms for classical analysis of large order control systems. The subroutines can support time domain and frequency domain analysis of both continuous and discrete control systems, with the major emphasis on sampled analysis. SAMSAN contains subroutines for 1) performing eigen analysis; 2) computing the zeros of transfer functions; 3) performing frequency domain computations, and root locus plotting in three domains S, Z, and W; 4) reducing the order of dynamic models through block diagonalization techniques; 5) computing matrix exponentiation; 6) numerically linearizing a nonlinear dynamic model; and 7) performing numerical integration.

The SAMSAN routines are used by a designer/analyst to construct a driver program for performing the particular analysis required. The SAMSAN driver

program can be linked with the standard IAC utility routines which store and extract data from the database. The designer therefore can utilize all of the IAC data handling capabilities, and also access data previously stored in the database by other IAC modules.

MODEL—The MODEL (Multi Optimal Differential Equation Language) program computes and plots time responses of linear and nonlinear continous control systems. The program takes the dynamic equations of the control system and vehicle model written in an easy-to-use higher level language format and translates these equations into a FORTRAN program for computing the time response. MODEL can be run in either an interactive or batch mode. It generates time response plots emulating a strip chart recorder and can plot directly on the user's terminal. Interactive operation allows the user to interrupt the program and change parameters and/or initial conditions, and then to restart or continue the computation as desired.

The IAC implementation of MODEL allows multi-user jobs on the same account. The IAC run command provides a convenient method of accessing the full range of MODEL capabilities during the precompile phase, the execution phase and the interactive phase.

DISCOS—DISCOS is a very general system dynamics module for time-domain and frequency-domain analysis. It analyzes multiple-flexible-body spacecraft, with the flexible body effects represented via elastic modes. Nonlinear dynamic and control effects can be handled, including large-angle Euler rotations between bodies. Frequency-domain analysis is performed via a linearization of the system equations. A capability has also been developed within IAC to provide DISCOS with input time varying quasi-static loadings, of either thermal or mechanical origin, for time-domain analysis. DISCOS is capable of obtaining its input matrices from an IAC database. DISCOS graphics capabilities are available within IAC, for display of time-domain variable relationships.

IAC Control Data Flow—IAC provides convenient data flow between control system modules and other modules of both the structures and thermal

disciplines. Figure 3-4 is a top level data flow schematic for control and structural interaction studies. The DISCOS program is structured to solve dynamics of multiply connected flexible bodies. IAC provides data flow from a NASTRAN definition of the flexible characteristics to achieve a complete description of the multi-body flexible dynamic system. A DISCOS frequency domain analysis can be performed to create a linearized model of the coupled flexible bodies. This model can be stored in an IAC database, in the form of matrices with associated labels, to aid the user in model interpretation and manipulation for control analysis studies. The model data can also be accessed by ORACLS and SAMSAN so that modern and classical control system design and synthesis can be done. Control system dynamics descriptions can be passed between ORACLS and SAMSAN via the database so that both modern optimal and classical analysis can be performed on the same control system, with convenient exchange of control system dynamic models.

3.4 ANALYSIS ENHANCEMENTS

In addition to the basic solution paths and the thermal, structural and controls modules already discussed, IAC provides several features which enhance the general analysis capabilities. These features include the MIMIC module; the EPIC module-implementation and support tools; several IAC data flow techniques; and the set of standard IAC utilities.

MIMIC—A significant enhancement to various analyses is provided by the model transformation and interpolation capabilities of the MIMIC module. MIMIC transforms field values (e.g., scalar temperatures or pressures, or nonscalar stress tensors) from one model to another, using a linear interpolation scheme. MIMIC can also be used as a very general multi-coordinate interpolation capability for array type data. Used as a spatial interpolator, MIMIC provides coordinate based interpolation over 1-D, 2-D or 3-D metric space. Used in a more general fashion, MIMIC can provide interpolation over time or an arbitrary set of other parameters.

MIMIC provides a user accessible interface between the calculation of interpolation coefficients and the use of these coefficients in transforming field variable data. The coefficients are stored in the form of an IAC data-structure, which can be queried or modified, if necessary, by a user. For example, a user might find it appropriate to develop these coefficients based on an algorithm other than continuous linear interpolation, or to modify some coefficients in the area of an internal insulated boundary. Once satisfactory coefficients are obtained, they might be reused to transform field variable data for several different analysis cases.

EPIC—Considerable emphasis has been given to easing the task of incorporating new analysis modules into the IAC system. IAC experience with a variety of different software packages and operating environments has resulted in development of several features which aid the module implementation, and which enhance the operation of a module over that obtainable outside of the IAC system. These features were considered important enough to warrant the codification of the acronym EPIC (Enhanced Program Interface Capability) to

collectively describe the concepts and software tools provided for this purpose.

Implementation of a module within IAC is accomplished via definition of appropriate tables, and without any modification of the executive program. An entry is created in a module list table; this includes overall user help and examples information pertinent to the module, and defines the location of the module software. A module run table is also created; this table defines characteristics, rules, help information, etc., for each run-time parameter which may be specified by the user. The EPIC software facilitates programmer access to these parameters, within the module job control language or within the executable code.

In addition to dealing with the above described tabular information and job control language, the implementation of a module may involve establishing appropriate technical data flow between the module and the IAC system. Alternative techniques provided by IAC for accomplishing such data flow are summarized in the next topic.

Standalone modules typically evolve from a specific-application single-user operating environment, and the technical evolution is not often matched by corresponding improvements in the user interface and data handling methods. For example, various types of file conflicts can occur between different users executing the same module, or between batch and interactive tasks performed by the same user. An EPIC concept is to provide for generation of unique specifications for module associated files. In addition, module scratch files may automatically be assigned to a system provided scratch storage area, where large short-term storage requirements are shared and do not impact the individual user. Convenient user options are also provided for disposition of output files (e.g., immediate printing, storing, or holding in a print queue).

The capability for generating user-specific versions of the executable module is provided for many analysis programs; in this case the IAC system relieves

the user of concern for what routines and libraries must be accessed in order to link with the special user provided routines.

Another EPIC concept is to provide a general-purpose USER module to enable basic execution, within the IAC system, of an arbitrary module or procedure. This capability is useful for new software development, and for handling special-purpose user-owned modules, without any system related modifications.

EPIC also provides some standard programmer-callable job control procedures and utility routines which are supplied as part of the IAC system software. These might, for example, comprise the total set of job control language for certain types of modules. They might handle the retrieving and interpreting of the module parameters available from the executive program. Or they could enable the application programmer to record and process error messages in a standard manner.

Data Flow Techniques—IAC facilitates the flow of data between different modules or between a module and the user, by providing a central database storage area, standard data structures and formats, and various data management tools. In order to serve the needs of diverse modules and users for communicating with the database, IAC provides three different data flow techniques. Referring to Figure 3-5, these techniques are denoted respectively by the terms "integrated," "interfaced," and "generic." Each technique is useful for certain types of applications, depending upon the particular module characteristics, the resources available, and current and future user needs.

In the integrated approach, IAC database read/write utilities are linked directly into the module, along with any necessary code for conversion of data structures and formats. This approach can be effective if a fairly mature module source code is available, programming personnel are familiar with the code, and the current and future data flow requirements are well defined. Integrated data flow has been provided for IAC modules such as DISCOS and ORACLS.

The interfaced technique uses an intermediate module, called an interface module, to bridge between the database and a module-formatted binary file. (In some cases, especially for database-to-module flow, a character formatted file such as a module card image input file may be utilized.) This technique can be used effectively where module source code is not available or the software is difficult to modify, and the module is capable of reading or writing the required data via a module defined file. It is also helpful where users wish to save significant quantities of module computed data on an output file, and then to decide at some later time which selected items of data are to be converted and stored in the database. Interface modules are used to provide data flow for IAC modules such as SINDA and NASTRAN.

The generic data flow approach is characterized by a human-readable character formatted file, and user-in-the-loop control. The process tends to be less automated than for the integrated or interfaced technique, but provides considerable flexibility and may in some cases prove to be more efficient. The IAC generic data flow capability is used to extract and query data from printed output files, and to generate partial input files. The generic technique has many other uses as well, such as in transferring data between two different types of host computers.

Currently, most of the standard IAC modules and solution path operations are based on either the integrated or interfaced technique. The generic technique, however, has been found to be of considerable utility, and the IAC user group has been especially enthusiastic in its support of generic data flow capabilities.

IAC Utilities—The IAC utilities are callable routines which provide a foundation for development of system software, and for interfacing of application modules. Approximately thirty IAC utilities are provided, involving five basic functional areas.

The IAC data transfer utilities move data between different storage areas; for example, between a database and a workspace, or between a workspace and a host directory.

The database and file management utilities handle the opening and closing of a database, concurrent database access, data cataloging and archiving, and delete/purge/rename type operations.

Workspace utilities provide for the creation and management of dynamic memory, including operations such as memory area allocation and deallocation, extension and truncation, location or retrieval of area characteristics, etc.; this function is essential for effective support of modern computational and query processes.

A number of general purpose utilities provide miscellaneous support functions such as initiation and termination of communication with other utilities, parameter file reading and writing, and character array/string manipulation.

Finally, the message utilities provide standard support for recording and displaying messages (e.g., information, warning and error), for error recovery, and for debugging new software.

4.0 EXECUTIVE AND DATA MANAGEMENT

The IAC executive establishes a unified interface for all of the system capabilities. It provides a dual command/tutorial language with extensive user help and examples; a dynamic user workspace for processing data; a capability for executing other modules with user specified parameters; detailed data manipulation and query; and a convenient interface to all host operating system commands.

The IAC data management system provides several standard data structures and corresponding query capabilities. It also provides a very general database which stores and catalogs the IAC-defined data structures as well as arbitrary application-module defined files. The IAC data management approach thus combines the advantages of standardization and query which are embodied in formal data management systems, with the critical need of most engineering computational modules for efficient handling of the module formatted files.

IAC Task Control—The IAC executive is designed to serve as either a master controller which drives other software, or as a slave which is driven by an application program. The executive may perform its tasks in either an interactive or a batch mode of operation.

A schematic of IAC task control, and the functional relationships between the executive and other system modules, is shown in Figure 4-1. There the upper and lower circles denote the master and slave functions, respectively, of the executive. The left and right sides of the figure respectively describe the interactive and batch IAC modes of operation.

An IAC task usually begins with the interactive master executive. Considering the interactive mode of operation, the executive might execute an interactive application module, which could in turn execute the executive program as a slave in order to perform application specified tasks. As an example, the ORACLS controller design/analysis module is typically executed by the executive with a user specified basic input file. ORACLS reads several plant/controller system definition matrices from the database, performs a

first iteration analysis, and stores numerical and graphical results in the database. The user then calls upon the slave executive to query the database and make appropriate updates to the system definition, after which control is returned to ORACLS. This iterative process is repeated until an acceptable design is obtained, after which final results are stored in the database and control is returned back to the original interactive executive.

The interactive executive may also submit a batch job, thus initiating the batch mode of operation. Note that all executive and application modules may communicate with one another via the common IAC database.

An important advantage of the IAC task control approach is that software functions are kept highly modular. All of the executive data query, etc., capabilities can be made available within an application module, without requiring that module to link in and integrate a large amount of foreign software.

User/Executive Interfacé—The ACE executive provides a dual command and tutorial prompting interface. A user controls the sequence of tasks within the executive program by giving either complete or partial commands, and by relying upon the command processor to prompt for any missing information. Thus the user can choose a mode of input consistent with experience level on a particular capability, and the executive can "adapt" to the appropriate type of communication with the user.

In order to support this dual interface, all ACE commands are defined by the system as a series of "nodes" (parts or sub-parts), and stored in a table type format. For each node the table defines an appropriate user prompt, user help information in the form of explanations and examples, permissible data input classes and values, and relationships between the given node and any lower level nodes. The command processor compares user input against the structure of the table, and transfers from node to node within the table as required. Whenever additional input is needed, the prompt associated with the corresponding node is displayed to the user. The processor provides appropriate node position transfers and recovery in case of input errors.

The dual interface concept supports a great deal of flexibility relative to the user's mode of input. The associated table driven command concept simplifies any desired alteration of prompts and help information to suit the needs or preferences of a particular user group.

Another significant feature of the ACE user interface is the capability to predefine and then reference an auxiliary sequence of commands or input items. This may be done via two methods. The first is to simply place the auxiliary input on a host file, and branch to this file during execution. The second method is to use the ACE macro substitution capability. Macros may be defined in IAC table (relation) form, and stored in a database. A macro reference may include specific run-time arguments, and results in macro text or data being inserted into the command sequence. Macros are especially useful for implementing analysis solution path sequences, where specific file names etc. need to be inserted at run time.

Database—A schematic of the IAC database is shown in Figure 4-2. Multi-user concurrent access to the database is supported by an activities file, which contains appropriate information (user name, access type, etc.) for each database file access currently in progress. This allows for the identification of, and protection against, user access conflicts which might occur.

The catalog contains characteristics of all other files in the database; it includes titles and keyword identifiers, as well as pointers to the data file itself and to any associated user defined textual information. Detailed catalog query capabilities provide an analyst with the ability to regain familiarity with existing data files, and to verify the correct identity of a file prior to its use in further computations.

As an illustration, a user could make a request to "list the file name, creation date, creating module name, and title; for all database files created by SMITH, after July 1982; which contain the keywords TRUSS, MODES, and/or CONFIGURATION2." Such query capabilities are also valuable in identifying groups of files, prior to performing operations such as copying, deleting, or renaming of the files.

The catalog supports an automatic versioning capability for all database files. Versioning encourages backup against data loss, promotes recovery from some types of analysis errors, and helps users to differentiate between obsolete and current data.

The database contains two types of cataloged files:

(1) "Unstructured files," i.e., files defined according to individual module formats. Examples are a NASTRAN checkpoint file, a DISCOS plot file, FORTRAN source or object code files, and the input card image or output printer files for any module. In order to maintain computational efficiency, these files are left in their original module defined formats, without any IAC modification or appending of additional information.

(2) "Structured files," i.e., files defined according to IAC standard formats. These are also referred to as IAC data structures, and provide a common basis for communication between different IAC modules or between a module and the executive. There are three classes of IAC data structures—ARRAY, RELATION and USER. These classes are summarized in the Data Structures topic below, and associated information is given in the topic on Data Manipulation and Query.

External Database Interface(s)—The IAC database serves a specialized function, i.e. engineering analysis. As a consequence, an IAC database tends to be small in comparison with a global database serving the multitude of needs of a major flight program. Yet, it is recognized that IAC must be capable of interfacing with a variety of external databases, regardless of their sizes or characteristics. The need for an external interfacing capability is designated in the IAC architecture diagram (Figure 2-2) by the block titled "External Database Interfaces."

The architecture of the IAC is adapted to providing this interface in several ways. One approach is via an interface whereby the user utilizes standard IAC utilities or executive commands to "call" information from an external

database. Another approach is via an interface whereby the user addresses an external database utilizing its native commands and "calls" information for input into the IAC database. Two good examples of the need for external interfacing involve CAD activities and system configuration databases. Other needs could be related to the data gathered during actual spacecraft/vehicle operation.

Data Structures—Figure 4-3 illustrates typical forms for the IAC ARRAY, RELATION and USER data structures.

The ARRAY is probably the most important IAC data structure. It is effective for both computational and query purposes, and is used in most of the IAC technical module interfaces. This structure is a generalization of vector and matrix type data, and can be considered to include the RELATION structure as a subset. An illustrative 2-dimensional ARRAY containing transient nodal temperature data is shown in the figure. The core of the general structure consists of one or more arrays of arbitrary order (e.g., a matrix may be represented as a second order ARRAY), and each dimension (index) may be of arbitrary size. In order to facilitate user query of the data, each index may have an associated table consisting of one or more labels. Each part of the structure (i.e., label or array) can be referenced by name or by index value. The ARRAY data structure is used to handle much of the data for IAC modules and solution paths, e.g., nodal temperature data, mode shapes, mass and stiffness matrices, displacement and stress data, and plant/controller definition matrices.

The RELATION data structure represents table type data. It may be of arbitrary size, as measured by its number of columns and number of rows. Each column can be referenced by name, and values may also be referenced by index (row number). A relation containing simple configuration type data is shown in Figure 4-3. For many applications the RELATION is inferior to the ARRAY, in terms of storage and computational efficiency, as well as query capabilities. However, IAC usage has shown the RELATION to be effective in representing certain types of management data (e.g., data on costs, schedules and personnel), in defining project configurations, and for numerically

representing certain types of graphics data. The relational form also can be useful in communications between different integrated systems, since other data forms can usually be converted into logically equivalent relational form.

The USER data structure provides a user with considerable flexibility in defining arbitrary groups of data and relationships between them. IAC facilitates handling of USER data by providing a standard means of organizing the user defined data groups, and of managing the associated data types, data group identifiers, and sizes.

Workspace—With the current availability of extremely large computer memory resources (both real and virtual) there is a corresponding need for effective programming techniques which can utilize these resources.

Memory storage requirements within a system such as IAC tend to involve many different data groups, and neither the number of such groups nor their individual sizes can be predetermined. The IAC approach to this problem is to assign a RAM storage area, and associated storage management software, to dynamically allocate the assigned memory into logical data areas as they are required. This storage capability, including the IAC provided management software, is referred to as the IAC workspace and is shown schematically in Figure 4-4.

The workspace supports many operations of the executive program including data manipulation and query. An IAC workspace may also be utilized by an application module, either as a simple dynamic buffer for data input and output, or in a more sophisticated manner to provide dynamic memory management for all data. The workspace manager is programmed in FORTRAN but it overcomes a deficiency of that language relative to some other modern languages, namely that FORTRAN does not directly provide for dynamic creation and deletion of data areas. The workspace supports data area operations including allocation, deletion, extension or truncation, and others. By utilizing the workspace concept, IAC has been able to efficiently eliminate virtually all of the arbitrary and annoying restrictions on individual sizes

(for example, the maximum number of conditions in a user query, the maximum size of an input command or an output logical data group, the maximum number of parts in a data structure, etc.). The workspace manager software is provided as a set of documented utility subroutines, which facilitates its utilization by application module programmers.

Data Manipulation and Query—Analysis oriented capabilities for data manipulation and query have tended to ellicit enthusiastic support from two sources—from management personnel because of obvious increases in productivity, and from engineering analysts because of the substantial elimination of previously required tedious and menial data handling tasks. IAC data manipulation and query are usually associated with direct processing of data by an interactive user. However, these operations can also be grouped into predefined sequences (either by an application program or a user) to automatically accomplish various data computation or display tasks. The data operations described here may be performed on the general ARRAY or RELATION data structures. Some of them may also be performed on the database catalog, or on character formatted host files. The following operations are provided by IAC.

1. Data creation and deletion - defining, loading and unloading.

2. Display - alphanumeric printing, graphical plotting.

3. Editing - Value changing, sorting, name or format redefining.

4. Data selection and combination - selecting, merging, partitioning, extracting.

5. File management - cataloging, archiving, purging, deleting.

6. File transformation - array/relation, relation/host file, etc.

7. Math operations - array addition, multiplication, etc.

8. Data query - logical (equal, less than, set membership, etc.), computational (sum, mean, deviation, etc.).

Much of the power inherent in a data manipulation and query system is derived from the supported data types and the capabilities for data selection and comparison. The following is a list of IAC defined data types.

1. Integer
2. Real - single or double precision
3. Complex - single or double precision
4. Logical
5. Character - fixed or variable length
6. Non-scalars (vectors, matrices, etc.) for above data types - fixed or variable dimensions.

IAC provides the following types of selection and comparison operations.

1. Logical comparison between named and variables and constants, e.g., "X greater than 100." or "TIME most equal to 2.5".

2. Logical comparison between different named variables, e.g., "Y most equal to Z" or NODE1 less than or equal to NODE2".

3. Maximum/minimum selection, e.g., "STIFFNESS equals maximum" or "X equals minimum subject to condition that NODE less than 50".

4. Set membership and list comparison, e.g., "KEYWORDS contains all of the substrings MASS, TRUSS, MARCH 1983" or "NODE in the set (25, 30, 71 through 90 incremented by 2, 511, 519, maximum)".

5.0 CONCLUSIONS AND RECOMMENDATIONS

Applications Experience—The specific capabilities of the IAC thermal/structural/controls software have been used for a variety of analysis applications within Boeing and NASA/GSFC, during the system development. Many of these analyses have been standalone applications involving thermal, statics, dynamics or controls modules. In these applications the thermal and structural analysts have been mainly interested in extracting output data from their modules, storing it in an IAC database, and then querying and displaying the data. The controls analysts have also utilized these capabilities, but perhaps more importantly have come to depend upon the IAC-provided array creation and manipulation tools. Controls modules often communicate data directly to and from an IAC database, and the design of many of the data manipulation and query tools has been driven by requirements of the controls technology.

Some IAC applications have involved coupled thermal/structural or structural/control analyses; for example, deformations of local high temperature structural components, or time/frequency response of space structures. In these coupled applications the structural analyst may need to obtain nodal temperatures (sometimes transient but usually steady state) in order to generate thermal loadings for a static deformation analysis. In almost all practical cases, the thermal/structural analysis involves different thermal and structural models, and requires a mesh interpolation procedure. The controls or system dynamics analyst is interested in obtaining normal modes and associated structural definitions from a structural dynamics analyzer. It is of interest to note that in our experience the thermal analysts have seldom required data results from the other technologies. However, they have benefitted in some cases from access to the structural model geometry, and from an improved interface between the radiation property calculations (e.g., view factors and exchange coefficients) and the later thermal conduction/convection/radiation analysis.

A more general application of IAC has involved engineering management requirements, i.e., handling of cost and personnel data. IAC query, graphics

and generic interface capabilities are being effectively used to allow managers to interface with and maintain visibility of large volumes of these data produced by other organizations.

Benefits—The IAC system has had several obvious impacts on the engineering analysis activities which it has supported. It has achieved an expanded and more rapid flow of data between different project technologies; a common ground for communication between engineers of different disciplines; an increase in task/time productivity; and a better working environment for the engineering analyst.

IAC can effectively support the analysis of complex structures which require a truly integrated data flow. By improving the productivity of common engineering analysis tasks, IAC can free the engineer to perform more innovative work. For skilled users on tasks which require considerable data manipulation, query and plotting, IAC may improve the engineer's productivity by an order of magnitude with increased reliability. We hope the reader will obtain the code and try it to substantiate the IAC benefits first hand.

Recommended Improvements—In the computer age we find that no useful computer code is ever truly complete. NASTRAN, DISCOS and SINDA, for example, are continuing to be improved in terms of either computational speed or new capability. IAC is no different. At this stage in its development, IAC requires the completion of several key technical capabilities and system related improvements. The recommended enhancements to IAC are given below. It is realized that both costs and schedules are impediments to some of this work. However, once users have demonstrated the IAC potential to management, support for software development is easier to obtain. This has been the case at BAC, where several IAC related tasks are currently underway within the IR&D program.

IAC recommended improvements fall into three general categories. First, improve the existing IAC capability. These improvements would increase the IAC operating speed and/or increase the productivity of the IAC users. Second, add automated model generation features, data output processing, and

improved model and data graphical display. Also, an IAC/CAD interface has been explored and is beginning to evolve. Third, extend the technical capabilities within the structures, controls and thermal areas of Level 1. For example, include an optimization package, and incorporate new technologies such as optical performance analysis and design, RF antenna performance, power system analysis and aerodynamics.

These three areas are not necessarily prioritized or complete. They are the result of general BAC experience with the IAC, and include specific items to improve the ability of IAC to perform efficient analysis of space systems.

Within the thermal technology, enhancements to the thermal modeling, and structural/thermal model interfaces, are needed. Software is being developed to allow conversion of a finite element model (such as that generated by SUPERTAB) to a nodal network model (used by SINDA, etc.). A similar task has been planned to provide conversion of a finite element model to a radiation analysis model such as that used by TRASYS. These activities have a significant payoff, because of the large amount of sophisticated software already available for finite element model generation and display of associated field variable data.

In the area of IAC system related improvements, there is a need to further enhance the command language and the list of available commands. Improvements can be made in the processing of user errors. The set of approximately fifty currently available executive commands should be extended, especially in the handling of array data computation and manipulation. Graphics capabilities must be extended, with emphasis on improved data flow between IAC and the major currently available codes for math modeling and field variable display. Finally, IAC should be implemented on additional computers, covering the range from the newer generation micros up to the super mainframe machines.

6.0 REFERENCES

1. "Integrated Analysis Capability for Large Space Systems," R. G. Vos et al, D180-26027-1 and 2, Phase I Final Report, June 1980.

2. "IAC Functional Specification," W. J. Walker, D180-26889-1, Boeing Aerospace Company, January 1982.

3. "Integrated Analysis Capability for Design of Large Space Systems," Presentation by W. J. Walker, 21st AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, "ay 1980, Seattle, WA.

4. "Integrated Analysis Capability," J. P. Young et al, 2nd Chautauqua on Productivity in Engineering and Design: The CAD Revolution, November 1982, Kiawah Island, SC.

5. "Development and Use of an Integrated Analysis Capability," R. G. Vos et al, 24th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, May 1983, Lake Tahoe, NV.

6. "IAC Level 1 User Manual," R. G. Vos et al, NASA CR-175300, also D180-28341, Boeing Aerospace Company, February 1984.

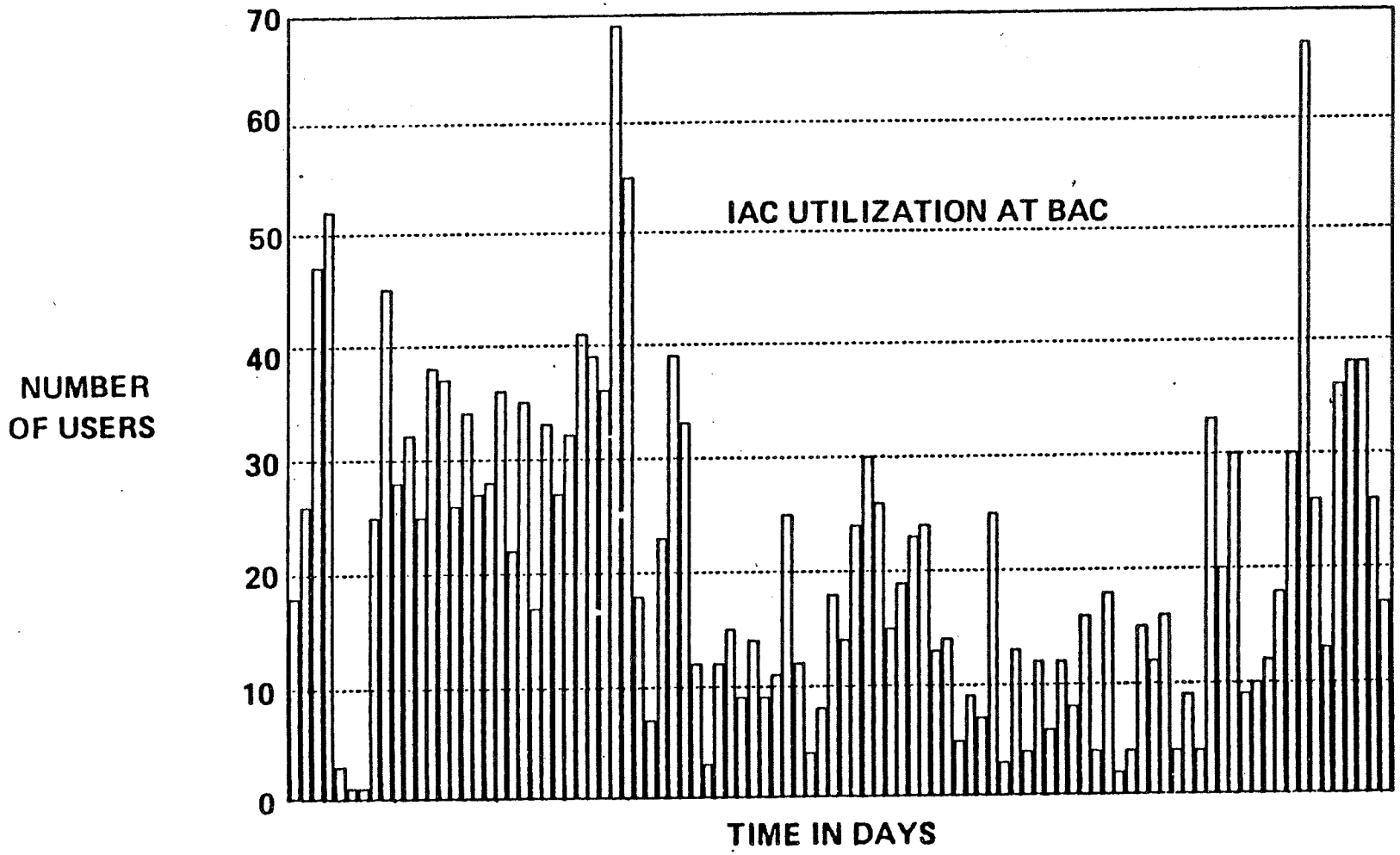
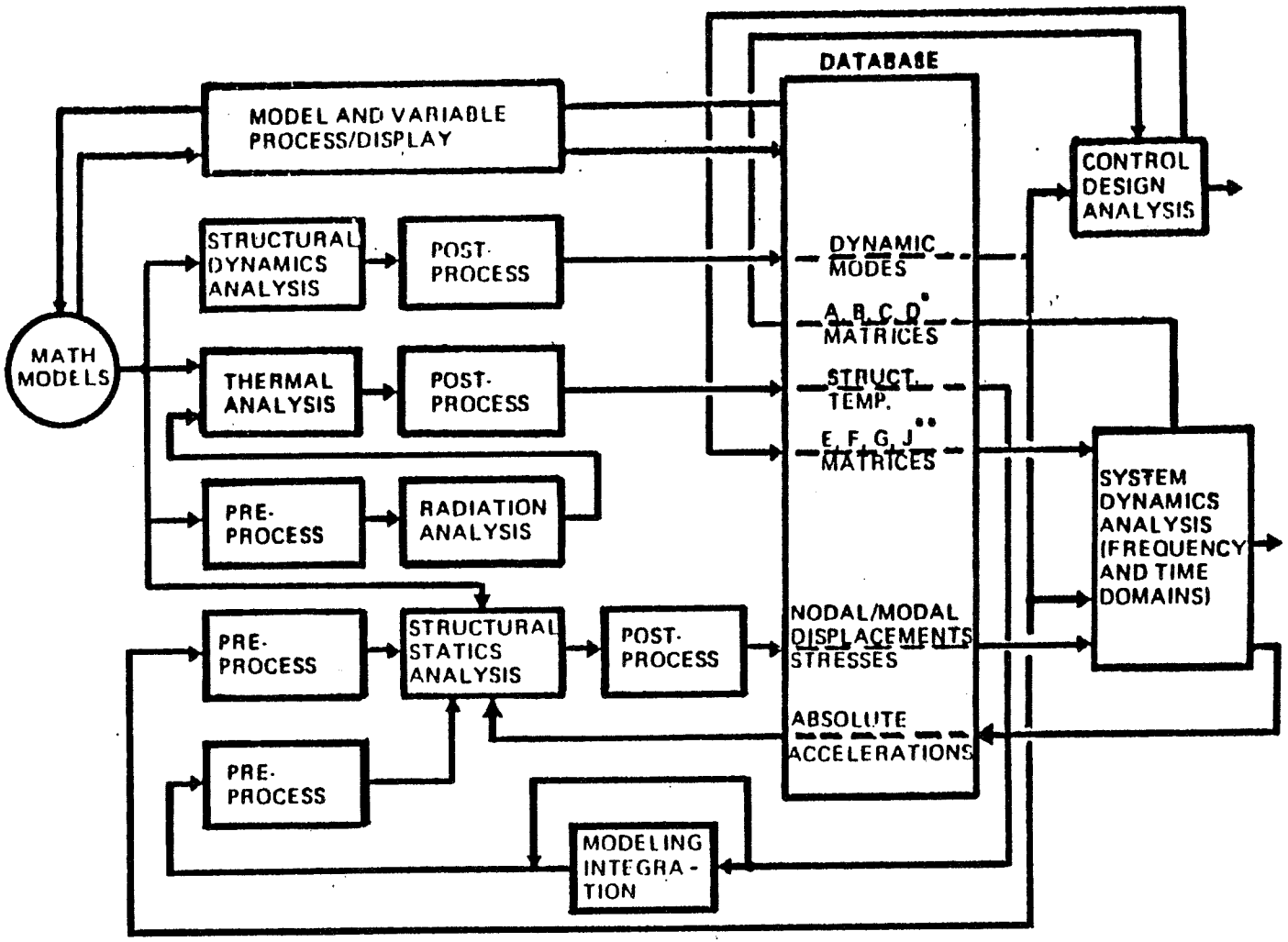


Figure 1-1: Typical IAC Utilization



*A,B,C,D = LINEARIZED PLANT (STRUCTURE) MODEL
**E,F,G,J = CONTROLLER DESIGN

Figure 1-2: IAC Data Flow Schematic

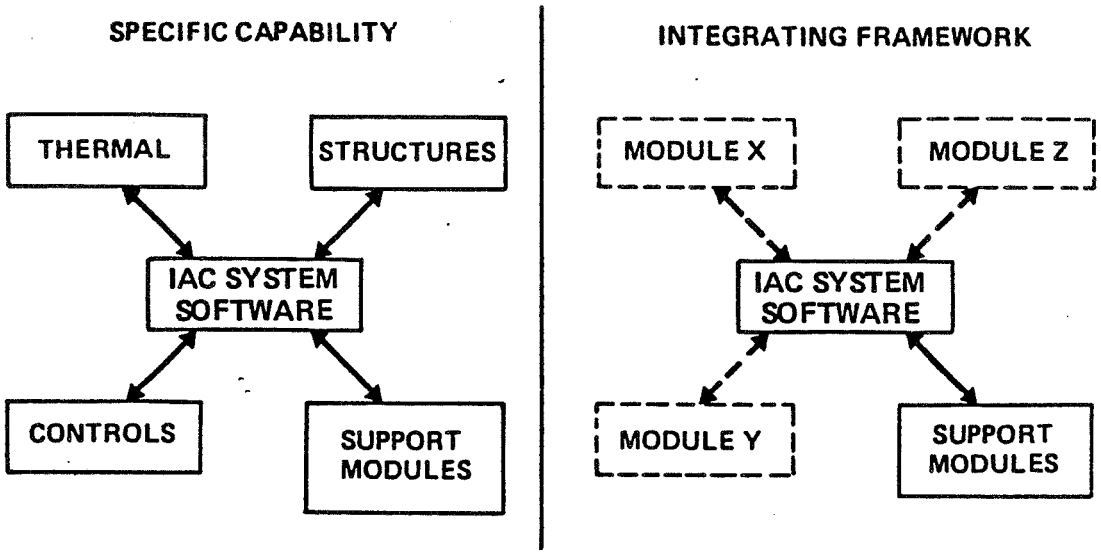


Figure 2-1: Two IAC Products

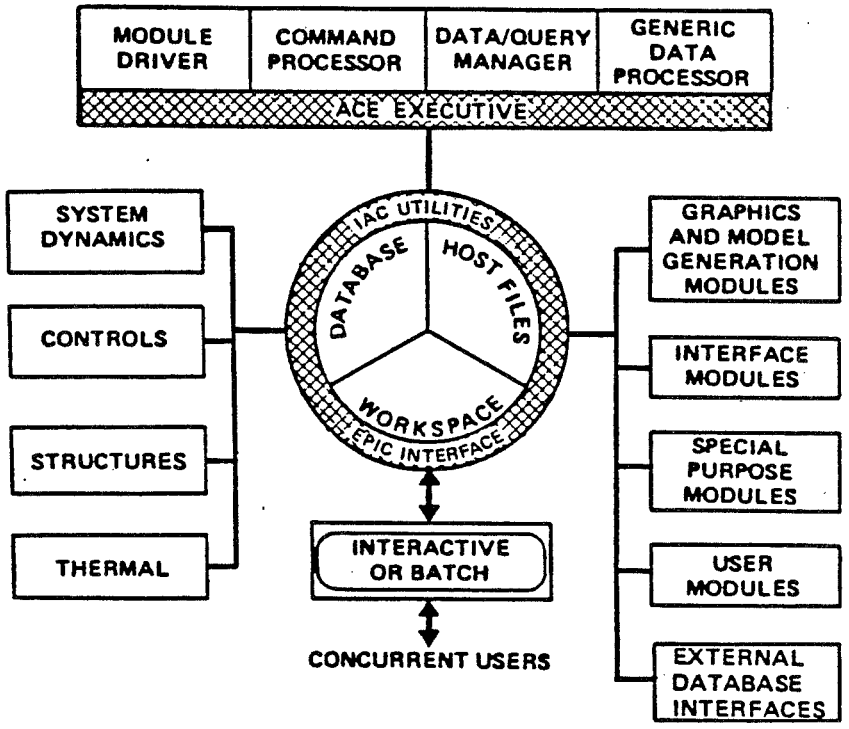


Figure 2-2: IAC Architecture Summary

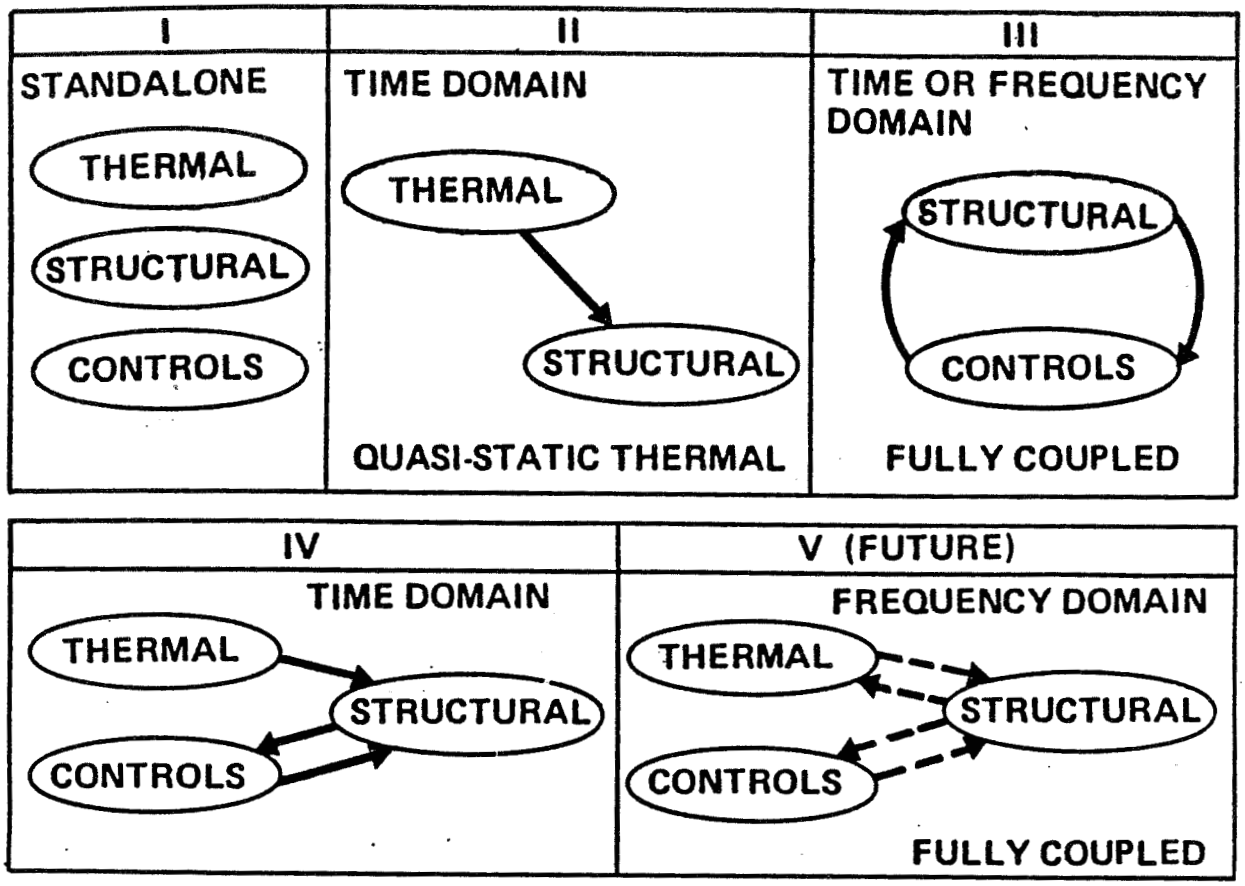


Figure 3-1: IAC Standard Solution Paths

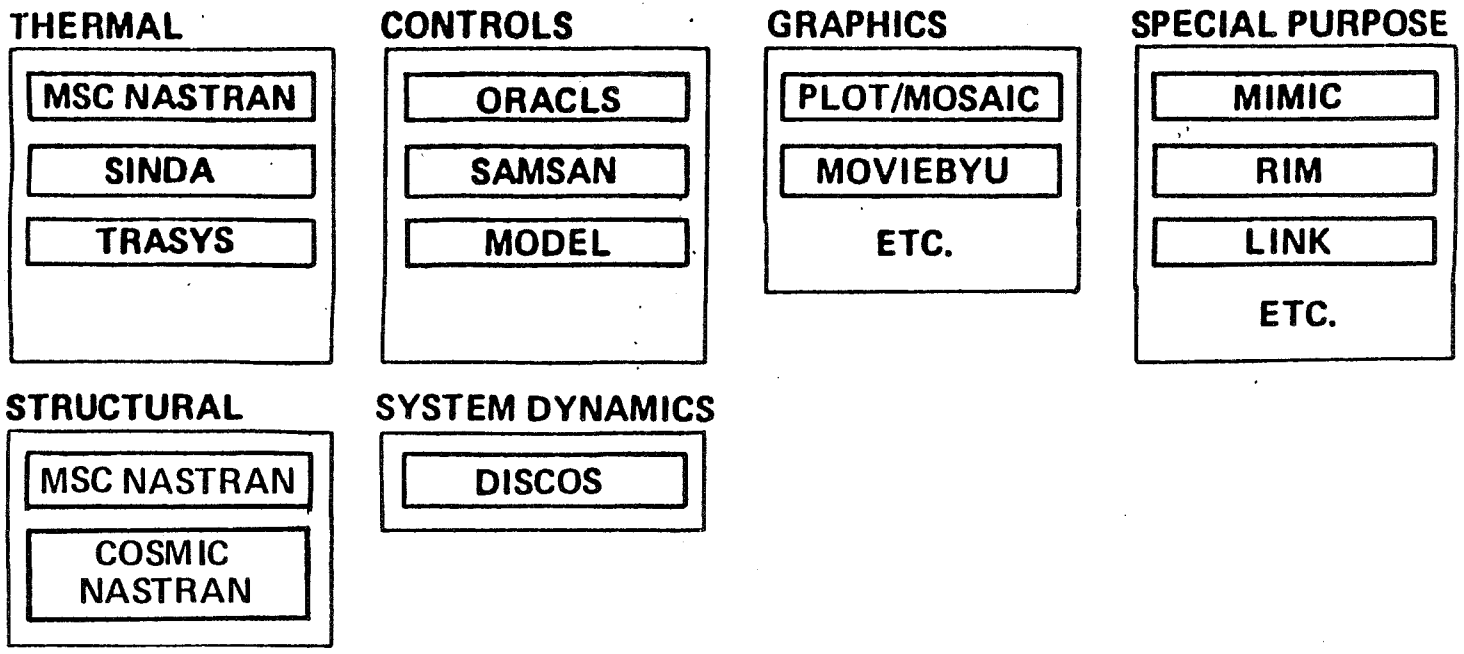


Figure 3-2: IAC Standard Application Modules

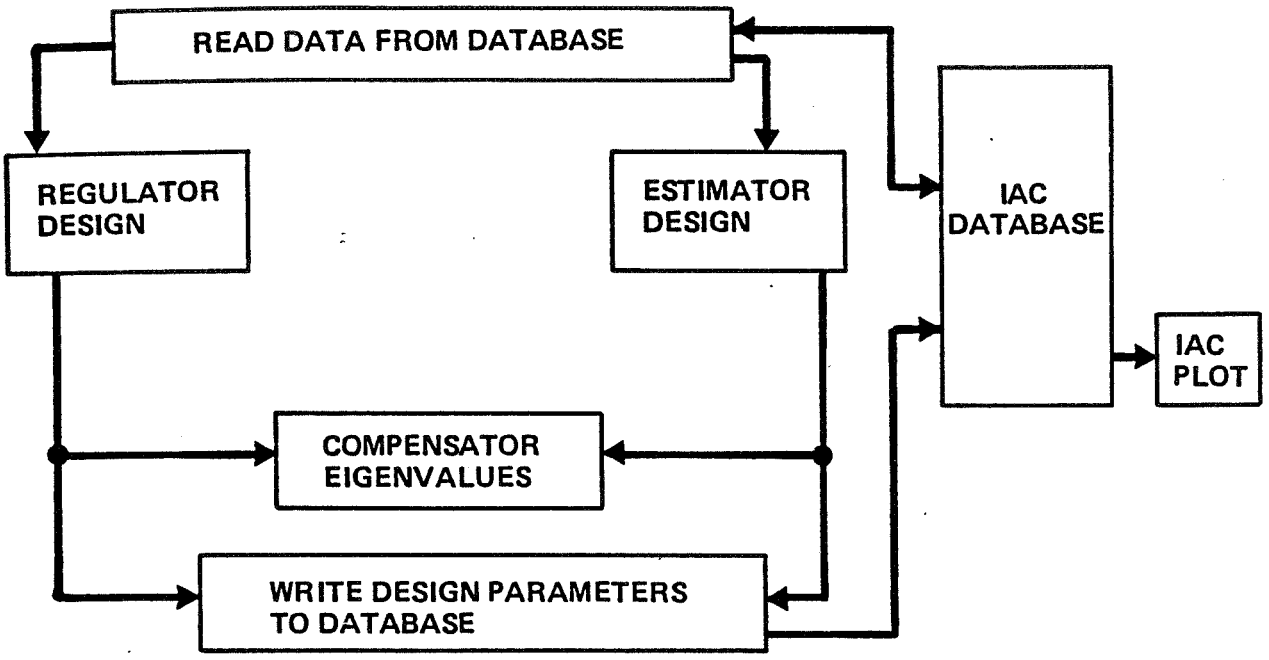


Figure 3-3: ORACLS Standard Driver REGEST

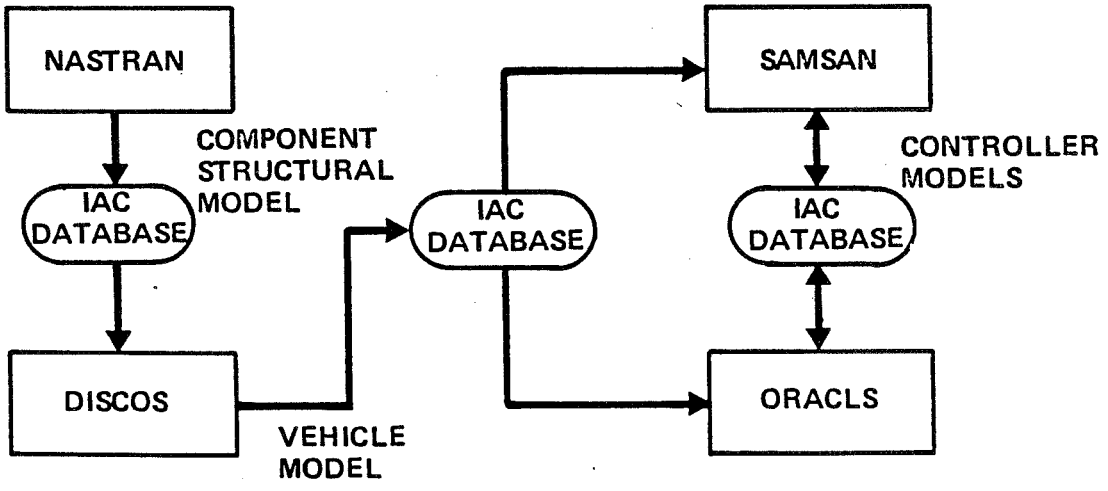


Figure 3-4: IAC Control Data Flow

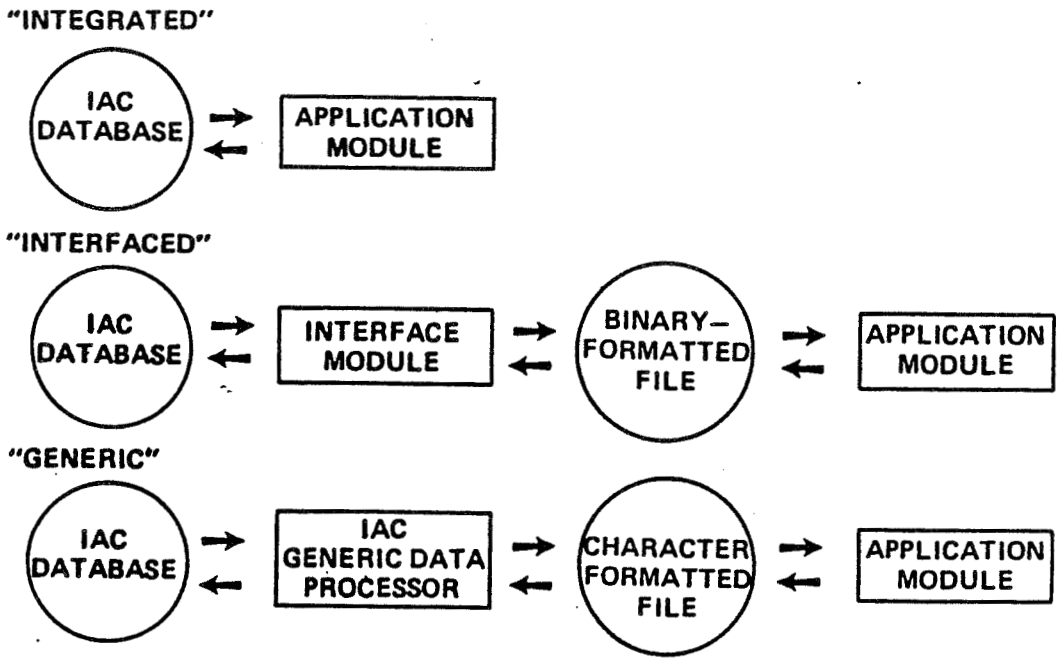


Figure 3-5: IAC Data Flow Techniques

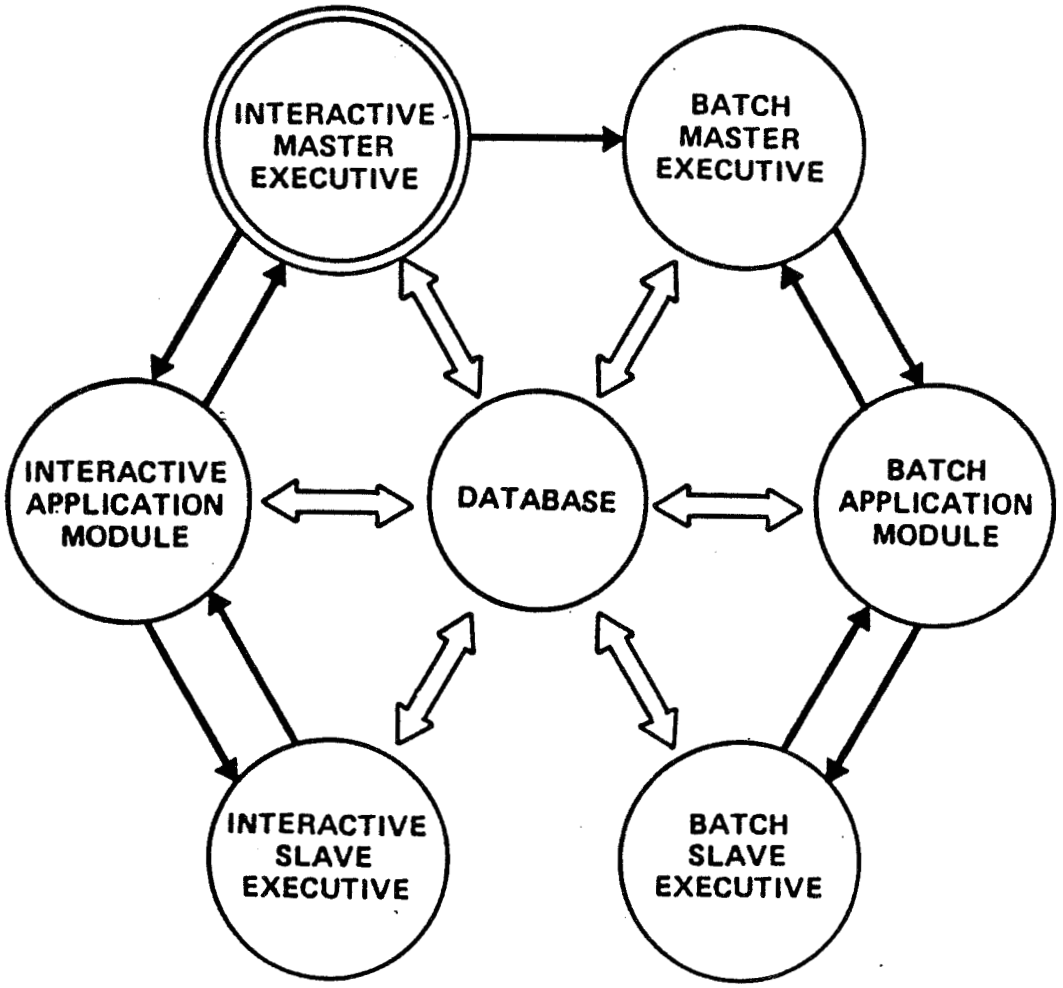


Figure 4-1: IAC Task Control Schematic

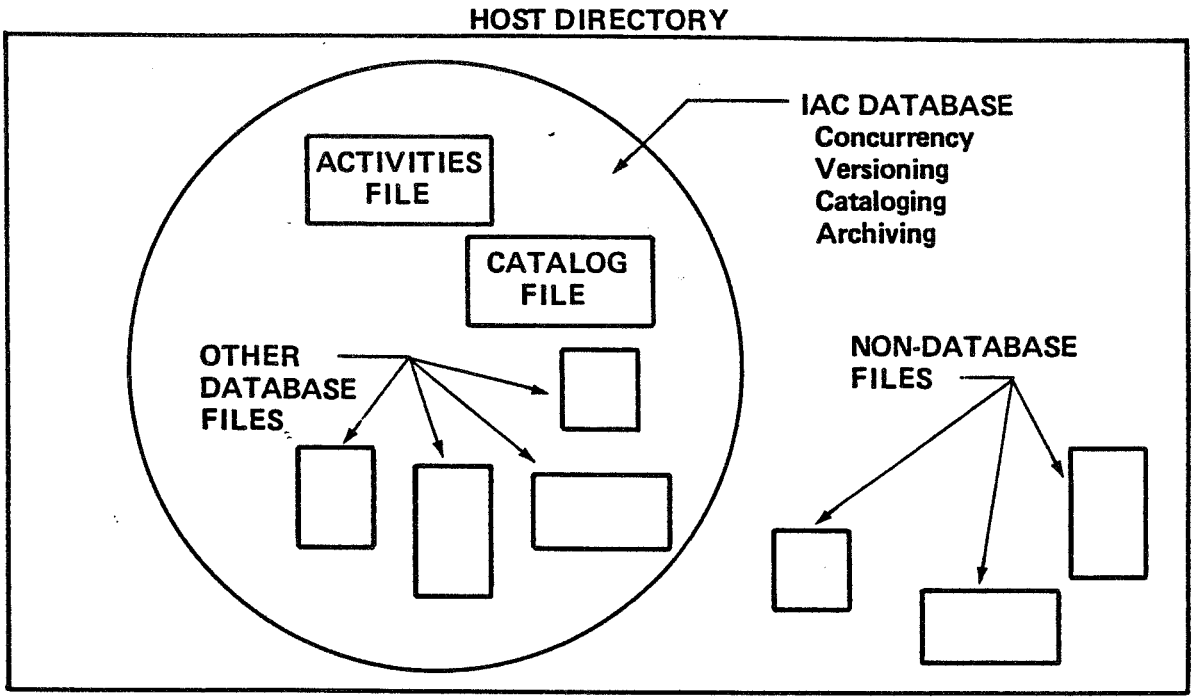


Figure 4-2: IAC Database Schematic

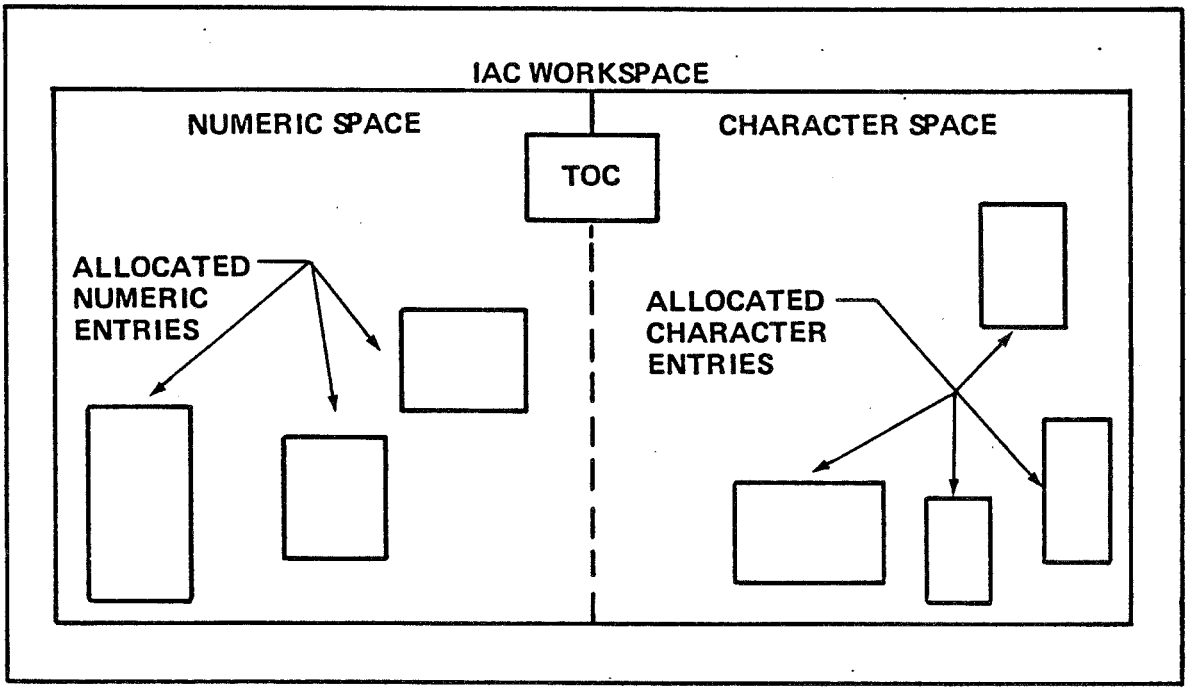


Figure 4-3: IAC Workspace Schematic

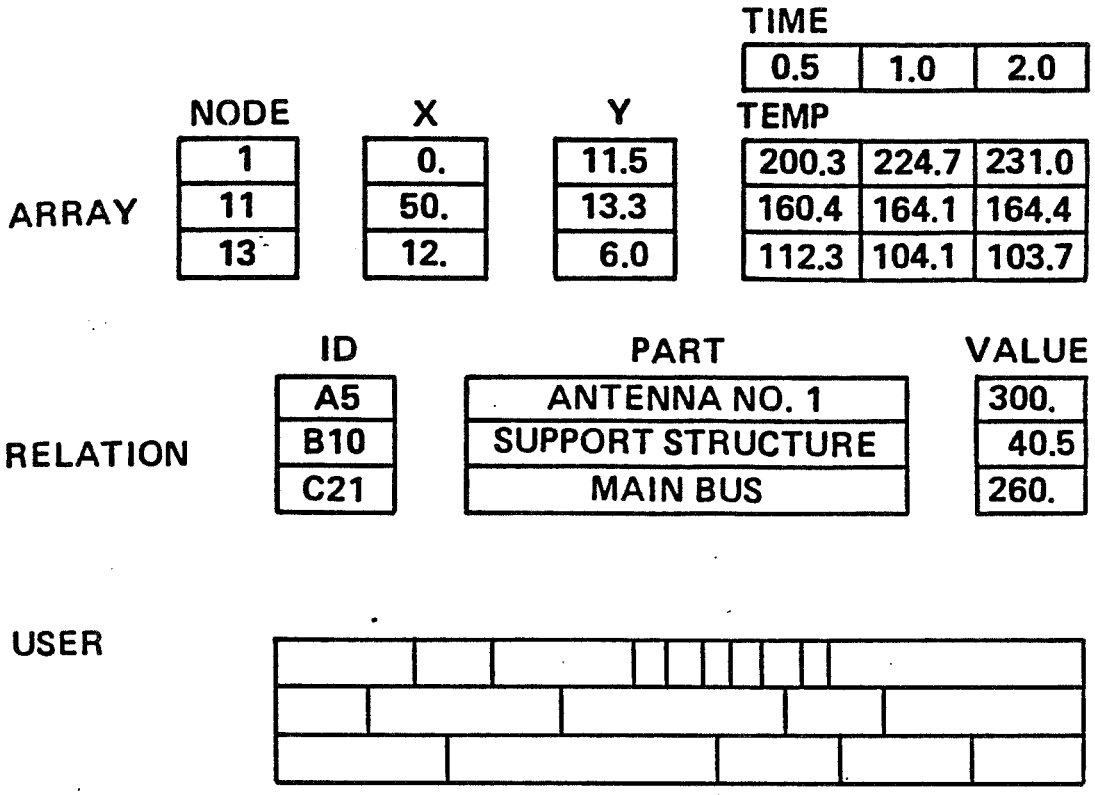


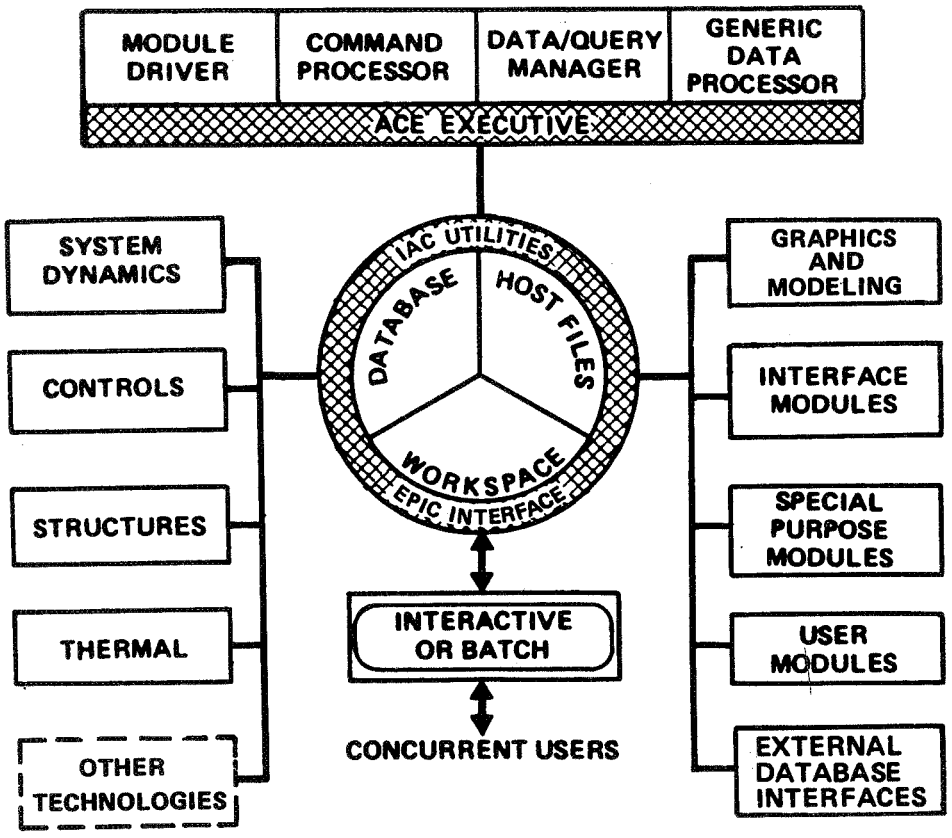
Figure 4-4: IAC Data Structures

COSMIC PROGRAM # GSC-12994

Integrated Analysis Capability

Volume 4 of 4

User Manual (Level 1.5)



Robert G. Vos, David L. Beste and Joseph Gregg

Boeing Aerospace Company
Seattle, Washington

Coordinated with
Goddard Space Flight Center
Greenbelt, Maryland

NASA
National Aeronautics and Space Administration

May 1985

THIS PAGE INTENTIONALLY LEFT BLANK

IAC USER MANUAL (LEVEL 1.5)
TABLE OF CONTENTS

		Page
Table of Contents		iii
List of Figures		vii
List of Tables		x
Introduction		1
Summary		3
IAC Report Form		8
1.0	Basic IAC Operation	1-1
2.0	Executive Commands	2-1
2.1	Command Diagrams	2-2
2.2	Macros and Amacros	2-128
3.0	Modules	3-1
3.1	MSC NASTRAN/TECKPLOT	3-2
3.2	DISCOS	3-10
3.3	TRASYS/TRASPLOT	3-14
3.4	SINDA/ISIN	3-18
3.5	ORACLS	3-25
3.6	MIMIC	3-28
3.7	INTA	3-34
3.8	INDA	3-39
3.9	INSAT	3-45
3.10	INSAM	3-49
3.11	INSA	3-53
3.12	ACE	3-60
3.13	PLOT	3-63
3.14	MODELI	3-65
3.15	USER	3-68
3.16	RIM	3-73
3.17	SAMSAN	3-78

		Page
4.0	Solution Paths	4-1
4.1	Path I - Standalone	4-3
4.2	Path II - Thermal/Structural	4-3
4.3	Path III - Structural/Control	4-5
4.4	Path IV - Thermal/Structural/Control	4-7
5.0	Data Organization and Storage	5-1
5.1	Data Areas	5-1
5.2	IAC Files	5-11
5.3	Structured Data Files	5-13
5.4	Relation and Array Files	5-19
5.5	Data Flow Techniques	5-22
6.0	IAC Utilities	6-1
6.1	Workspace Utilities	6-3
6.2	Database/File Management Utilities	6-10
6.3	File Transfer Utilities	6-16
6.4	Utility Condition Processing	6-26
6.5	Miscellaneous IAC Utilities	6-28
6.6	Utility Summary Reference	6-36
7.0	Module Implementation	7-1
8.0	References	8-1

		Page
Appendix A	- New User's Introduction To ACE	A-1
A.1	- How ACE Works	A-1
A.2	- ACE Terminology	A-3
A.3	- What Follows	A-5
A.4	- A Painlesss Introduction To ACE Commands	A-6
A.5	- Explanation of ACE Control Commands	A-13
A.6	- Help and Examples For ACE	A-15
Appendix B	- Plot Information	B-1
B.1	- Plot File Definition	B-1
Appendix C	- DISCOS Enhancements	C-1
C.1	- Nametag/ID Input-File Layout	C-2
C.2	- A, B, C Matrics	C-20
C.3	- Graphics	C-25
Appendix D	- ORACLS Implementation	D-1
D.1	- Standard Drivers	D-1
D.2	- User-Defined Drivers	D-34
D.3	- New IAC ORACLS Utility Procedures	D-35
D.4	- Modification And Operation Of Original ORACLS Package	D-38
Appendix E	- MIMIC Implementation	E-1
E.1	- Interpolation Theory	E-1
E.2	- Example Problems	E-5

		Page
Appendix F	- IAC Solution Path IV Methodology	F-1
F.1	- Theory	F-1
F.2	- Implementation	F-3
F.3	- Example	F-7
F.4	- Conclusions	F-12

IAC USER MANUAL (LEVEL 1.5)

LIST OF FIGURES

Figure No.	Title	Page
1.0-1	IAC Task Control Schematic	1-2
3.1-1	MSC NASTRAN/TECKPLOT RUN Schematic	3-3
3.2-1	DISCOS RUN Schematic	3-11
3.3-1	TRASYS/TRASPLOT RUN Schematic	3-15
3.4-1	SINDA/ISIN RUN Schematic	3-19
3.4-2	SINDA/IAC Arrays	3-23
3.5-1	ORACLS RUN Schematic	3-26
3.6-1	MIMIC RUN Schematic	3-29
3.7-1	INTA RUN Schematic	3-35
3.7-2	NASTRAN-Thermal/IAC Arrays	3-37
3.8-1	INDA RUN Schematic	3-40
3.8-2	NASTRAN-Dynamics/IAC Files	3-42
3.9-1	INSAT RUN Schematic	3-46
3.10-1	INSAM RUN Schematic	3-50
3.11-1	INSA RUN Schematic	3-54
3.11-2	NASTRAN-Statics/IAC Files	3-57
3.12-1	ACE RUN Schematic	3-61
3.13-1	PLOT RUN Schematic	3-64
3.14-1	MODELI RUN Schematic	3-66
3.15-1	USER RUN Schematic	3-69
3.16-1	RIM RUN Schematic	3-74
3.17-1	SAMSAN RUN Schematic	3-79
4.0-1	IAC Solution Paths	4-2
4.2-1	Solution Path II - Thermal/Structural	4-4
4.3-1	Solution Path III - Structural/Control	4-6
4.4-1	Solution Path IV - Thermal/Structural/Control	4-9

IAC USER MANUAL (LEVEL 1.5)

LIST OF FIGURES

Figure No.	Title	Page
5.1-1	IAC Database Schematic	5-4
5.1-2	IAC Workspace Schematic	5-4
5.1-3	Workspace TOC	5-7
5.3-1	IAC Data Structures	5-14
5.5-1	IAC Data Flow Techniques	5-23
7.0-1	Typical IAC Directory Organization	7-2
7.0-2	Module Execution Schematic	7-3
C.1-1	DISCOS Example Input File	C-5
C.2-1	DISCOS/IAC A, B, C Arrays	C-22
C.3-1	DISCOS Graphics Menu	C-29
C.3-2	DISCOS Typical Graphics Session	C-36
D.1-1	Example REGEST Input File	D-4
D.1-2	Example REGEST Run - Interactive Display	D-6
D.1-3	Example REGEST Run - Output File	D-12
D.1-4	Example TIMFST Input File	D-15
D.1-5	Equations for TIMFST	D-17
D.1-6	TIMFST Block Diagram	D-18
D.1-7	Definitions of TIMFST Compensator Matrices	D-19
D.1-8	TIMFST Debug Printout	D-20
D.1-9	Example TIMFST Run - Interactive Display	D-22
D.1-10	Example TIMFST Run - Output File	D-26
D.1-11	Example TIMFST Run - Plot	D-33
E.1-1	MIMIC 2-D Spatial Interpolation	E-2
E.1-2	MIMIC 3-D Spatial Interpolation	E-2
E.2-1	Meshes for MIMIC 2-D Example	E-7
E.2-2	Mesh Coordinates for MIMIC 2-D Example	E-7
E.2-3	Geometry for MIMIC 3-D Example	E-9

LIST OF FIGURES

Figure No.	Title	Page
E.2-4	Mesh Points for MIMIC 3-D Example	E-9
E.2-5	Mesh Coordinates for MIMIC 3-D Example	E-10
F.3-1	Solution Path IV Cantilever Beam Example	F-7

IAC USER MANUAL (LEVEL 1.5)
LIST OF TABLES

Table No.	Title	Page
2.1-1	Command Node Classes and Types	2-7
6.3-1	Qualifiers for File Transfer Utilities	6-19
6.3-2	ETABLE Format for IACREA and IACWRI Utilities	6-23
6.5-1	Parameter Card Image Format	6-32
6.5-2	Parameter Standard Types and Destinations	6-32
7.0-1	GEN Type Parameters	7-3
C.2-1	A, B, C Matrix Labeling Attributes	C-23
C.3-1	DISCOS X-Y Plot Variables	C-28
E.2-1	Field Values for MIMIC 2-D Example	E-8
E.2-2	Field Values for MIMIC 3-D Example	E-8

INTRODUCTION

IAC (may be pronounced I-ak) is an acronym for the Integrated Analysis Capability. This document is the Boeing developed User Manual for IAC Level 1.5, which is the first level of IAC made available for general public usage. Level 1.5 includes the software delivered by Boeing to NASA/GSFC in July, 1984, under contract NAS5-25767; it also includes the results of later coordinated development efforts by NASA and Boeing. References 1 through 6 (see Section 8) provide additional information on the IAC system.

You should find the current IAC system especially helpful in the thermal, structures, controls and system dynamics technologies. Although its development has been influenced by the requirements for design/analysis of large space systems, it has general applicability to problems in engineering, and in management of data and software.

The IAC Level 2 system is planned for release during mid-1986. It will provide additional executive and data management capabilities, a generalized CAD interface, and extended module interfaces.

Architecture - The IAC Level 1.5 architecture is depicted by the logo at the front of the manual. IAC combines major computer programs used in diverse analytical disciplines into a single, easily manageable package. The ACE executive module shown at the top of the logo manages the user interface, module executions, and alphanumeric and graphics data handling tasks. Communication and task-control transfer between modules are facilitated by the standard IAC utilities and EPIC interface software represented at the center of the logo. This software also facilitates access to, and data transfer between, an IAC workspace, an IAC database, the host file system and external databases. The external database interfaces block in the logo recognizes the need for IAC to communicate with other activities, e.g. via CAD databases or data gathered from actual spacecraft/vehicle operation. Data flow between the major technical modules is facilitated by the general executive capabilities and by specific module interface and modeling transformation software. The IAC computational and data handling sequences emphasize a modular engineer-in-the-loop approach; that is, the various capabilities have been provided as individual logical building blocks, to be

utilized by the analyst as required for a specific application. Both interactive and batch execution tasks can be accomplished within IAC, in a multi-concurrent-user mode of operation.

Use of the Manual - This User Manual is intended to be used in conjunction with existing documentation for the major IAC technical modules (see References in Section 8). In order to gain a cursory knowledge of IAC you should review the Summary, Sections 1 and 2, and Appendix A, along with information on specific modules or solution paths of interest. In order to gain a thorough understanding of the IAC system you may best proceed by reading the Summary and various sections of this manual in a sequential fashion, covering the basic IAC operation, executive commands, modules, coupled solution paths, data organization and storage, utilities, and module implementation, followed by the appendices covering details of executive and module operation.

Acknowledgements - We appreciate the contributions from a number of people to the IAC system development. Primary team members at GSFC have been

Joseph P. Young - Technical Monitor
Harold P. Frisch - Controls
Gary K. Jones - Structures
Joseph T. Skladany - Thermal

At BAC, the primary team members have been

William J. Walker - Program Manager
Robert G. Vos - Technical Leader
Elliot W. Brogren - Thermal
Gerald A. Price - Controls
David L. Beste - Lead Programmer
Joseph Gregg - Programming

In addition, a number of other people have contributed to the development and evaluation of concepts, software, and documentation. At GSFC, these include Joan A. Sanborn; and at BAC, Joseph A. Bossi, Richard M. Gates, Lauri H. Hillberg, Roy Ikegami, Robert E. Jones, John L. Tietze, and Suzanne M. Williams.

SUMMARY

The information in this manual is organized into seven major sections:

1. Basic IAC Operation
2. Executive Commands
3. Modules
4. Solution Paths
5. Data Organization and Storage
6. Utilities
7. Module Implementation

In addition, more detailed information on specific areas is provided by several appendices. Note that a report form is included following the Summary, for your convenience in communicating with members of the IAC development team. The major sections of the manual are summarized below.

Basic IAC Operation - IAC emphasizes an interactive mode of operation, although batch tasks may be easily accomplished as required. Section 1 describes how to execute IAC via the ACE executive module, and how to accomplish interactive and/or batch tasks. An explanation is given of the relationship between the ACE executive and other modules in the system. Data storage and access are then summarized, relative to IAC user workspaces, IAC databases, and the host file system.

Executive Commands - Section 2 covers the executive commands which are available to the user. The executive is controlled by a dual command language and tutorial prompting interface, which provides extensive interactive features including detailed on-line help and examples. The structure of the command language is described, and each command is described via pictorial diagram and facing-page text documentation. Also described are the ACE macro and amacro features, which support development of user defined command procedures.

Modules - The available IAC modules are documented in Section 3. IAC modules are individual executable program units, which are usually executed by, and communicate with, the ACE executive. IAC modules include major technical modules (e.g. NASTRAN, SINDA and DISCOS) which perform computational tasks; interface modules (e.g. INDA and ISIN) which provide required data-flow; and special purpose modules such as ACE and PLOT. Section 3 decribes run parameters, files, and basic operation for each module, along with associated flow diagrams. The following is a summary of the modules currently available within the standard IAC system.

1. MSC NASTRAN/TECKPLOT - A general purpose finite-element analysis program (and associated graphics package), with emphasis in the structural and thermal technologies.

2. DISCOS - A very general system dynamics module, for time-domain and frequency-domain analysis of multiple-flexible-body spacecraft.

3. TRASYS/TRASPLOT - A thermal radiation analysis module (and associated graphics package), which computes radiation loading and member exchange characteristics.

4. SINDA/ISIN - A general purpose finite difference thermal analyzer (and associated IAC interface module), which includes the SINFLO fluid flow addition.

5. ORACLS - A collection of subroutines for design of controllers and filters, emphasizing modern control methods.

6. MIMIC - a modeling integration and interpolation capability, e.g. for converting nodal temperatures or other field variables for one mesh to those for another mesh.

7. INTA - An interface module which stores computed data from the NASTRAN thermal analyzer into IAC database files.

8. INDA - An interface module which stores computed data from the NASTRAN normal-modes analyzer into IAC database files.

9. INSAT - An interface module which uses nodal temperature data in an IAC database array to automatically generate thermal loading input data for a NASTRAN static deformation analysis.

10. INSAM - An interface module which converts a NASTRAN static deformation model from a nodal basis to a modal basis; it uses mode-shapes data in an IAC database array to automatically generate NASTRAN input data required for this purpose.

11. INSA - An interface module which stores computed data from the NASTRAN statics analyzer into IAC database files.

12. ACE - The IAC executive module; it may serve either as a master controller for executing user commands and other modules, or as a slave which can be executed from within another module.

13. PLOT - An IAC plot module; it generates graphs and charts which display relationships between variables in an IAC file.

14. MODELI - a controls application module which generates and displays numerical solutions of ordinary differential equations.

15. USER - A generic IAC module interface, which provides a simplified approach for incorporating or testing new modules.

16. RIM - A standalone relational database mangement module; IAC provides an interface for transferring data between RIM and IAC databases.

17. SAMSAN - A collection of subroutines for classical controls analysis of large order systems, emphasizing sampled data.

Solution Paths - Some of the technical capabilities of IAC are described in terms of solution paths, which provide particular computational and data flow capabilities. Section 4 decribes modules and data flow, run parameters and

files, and flow diagrams for the major currently available solution paths. The following is a summary of these paths.

Standalone - The IAC standalone solution path is considered to be the uncoupled operation of each technology or major technical module, i.e. without significant data flow between modules. However, the module is supported by the IAC executive capabilities and it may communicate with an IAC database, in order to obtain user defined input data or to provide computed data for user evaluation.

Thermal/Structural - The basic form of this solution path uses an input thermal loading environment to compute steady-state or transient thermal deformations, via the coupling of a thermal analyzer and a structural statics analyzer. An IAC thermal/structural path could, for example, involve modules such as TRASYS to compute radiation loading and exchange characteristics; NASTRAN or SINDA to compute nodal temperatures; MIMIC to resolve any required modeling differences; INSAT to generate NASTRAN thermal loading data; and NASTRAN to compute static nodal displacements. The IAC executive and interface modules could be used at various stages to generate, transfer and evaluate required data.

Structural/Control - This solution path provides either a time-domain or frequency-domain analysis, via the coupling of a structural normal-modes analyzer and a system dynamics module. An IAC structural/control path could involve NASTRAN to compute mode shapes and other dynamic characterisitcs for each flexible body in a connected system; and DISCOS to perform a time-domain or frequency-domain analysis of the entire multiple-flexible-body system, including both structural and control effects. The IAC executive and interface modules could again be used during this process for handling of data, as appropriate.

Thermal/Structural/Control - This solution path provides a time-domain system dynamics analysis, including the effects of time varying but quasi-static thermal (and/or mechanical) loadings. The term quasi-static is used to indicate loadings which are weakly coupled with the system dynamics behavior, i.e. loadings which can be computed a priori without direct regard for the dynamic changes in configuration. An IAC thermal/structural/control path

could involve all of the modules in the thermal/structural path; INSAM in addition to INSAT in the thermal/structural computations to convert the computed displacements from a nodal basis to a modal basis; NASTRAN in the structural/control path; and DISCOS to perform a time-domain analysis of the multiple-flexible-body system, including input quasi-static modal displacement loadings determined from the thermal/structural solution.

Data Organization and Storage - Section 5 describes the data areas used by IAC; these include IAC workspaces, IAC databases and the host file system. The concept of IAC files is then discussed, including file types and access, and the cataloging of files. The organization and types of structured files are described, and in particular the definition and query of IAC relations and arrays.

Utilities - IAC utilities documented in Section 6 are provided for support of application programmers and users. The same utilities are used in the IAC system modules, including the ACE executive. Utilities are grouped into several functional categories; these include the workspace, database and file management, file transfer, condition processing, and miscellaneous support functions. Documentation includes related information on the workspace and database, and a description of each utility with its calling arguments.

Module Implementation - Section 7 provides information needed for implementing new modules, and for overall IAC system implementation and maintenance. A directory schematic is shown for the system software, and a flow diagram is given which illustrates the system operation and the execution of modules via the ACE executive. The module/run tables and the parameter file, involved in implementing a new module, are also documented.

THIS PAGE INTENTIONALLY LEFT BLANK

IAC REPORT FORM

Members of the IAC development team would welcome your comments, suggestions, or reporting of apparent errors in software or documentation. The reverse side of this page is a form provided for that purpose.

It is suggested that reports be sent to one of the following.

Harold P. Frisch
Code 712
NASA Goddard Space Flight Center
Greenbelt, MD 20771
Phone: (301) 344-8730

Robert G. Vos
Mail Stop 82-97
Boeing Aerospace Company
P. O. Box 3999
Seattle, WA 98124
Phone: (206) 773-9564

Name: Phone:

Title/Group:

Address:

Comment or suggestion ☐

Apparent error ☐

1.0 BASIC IAC OPERATION

This section of the User Manual describes IAC execution, interactive and batch tasks, and the relationship between the ACE executive and other modules in the system. Data storage and access are summarized, relative to the IAC workspace, IAC files and databases, and the host file system.

IAC Initiation - You will typically initiate IAC from an interactive terminal under control of the host operating system, by typing in the symbol "IAC". This causes the 'primary' ACE executive to begin execution, and to display the standard ACE prompt "ACE >". You may then enter any complete or partial ACE command, or an ACE control sequence. See Section 2 for a description of available commands.

Task Control - The ACE RUN command executes a module. The ACE SUBMIT command submits a batch job, which is managed via a run of a 'secondary' ACE executive, i.e. a new logical copy of ACE.

For each run of ACE or another IAC module, a 'runid' is generated (this is a unique 8-digit integer based on the host system clock). The runid is used for various purposes, usually associated with naming of temporary files for the run, e.g. a module parameter file of the form I12345678.PPP. Such files are usually deleted automatically by IAC at the end of the run, but you may notice them residing temporarily on your default directory. IAC automatically provides a single log file for all activities accomplished by a primary or secondary executive, e.g. for all module runs and host command executions accomplished. A new log file is created for each batch job submitted.

The ACE executive is designed to serve as either a master controller which drives other software, or as a slave which is driven by an application module. A schematic of IAC task control, and the functional relationships between ACE and other system modules, is shown in Figure 1.0-1. The double circle represents the primary ACE executive; the upper and lower circles denote the master and slave functions, respectively, of ACE; and the left and right sides of the figure respectively describe the interactive and batch modes of operation.

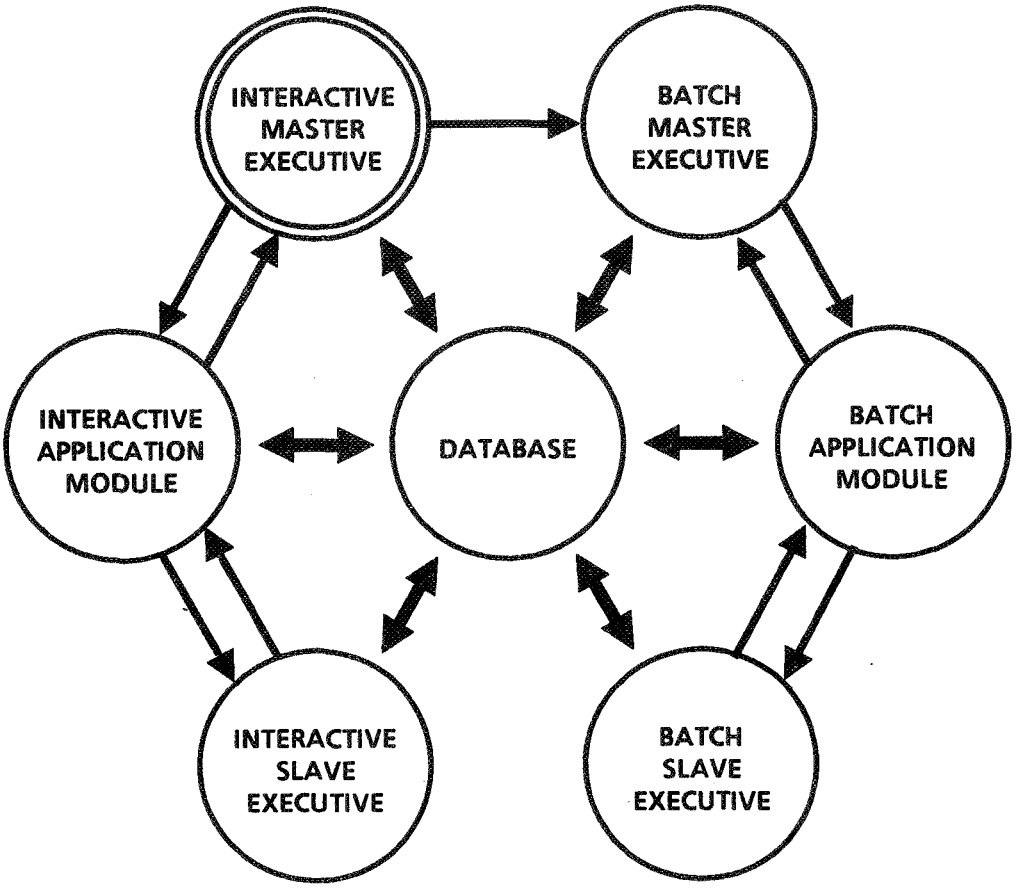


Figure 1.0-1: IAC Task Control Schematic

Considering the interactive mode of operation, the master ACE executive might run an interactive application module, which could in turn run ACE as a slave in order to perform application specified tasks. As an example, you might run the ORACLS controller design analysis module within ACE using a specified basic input file. ORACLS reads several plant/controller system definition matrices from the database, performs a first iteration analysis, and stores numerical and graphical results in the database. You may then call upon a slave ACE to query the database and make appropriate updates to the system definition, after which control is returned to ORACLS. This iterative process is repeated until an acceptable design is obtained, after which final results are stored in the database and control is returned back to the original master ACE.

Any ACE executive may also submit a batch job, thus initiating, for example, the batch mode of operation on the right side of the figure. Note that all modules may communicate with one another via the common IAC database.

An important advantage of the IAC task control approach is that software functions are kept highly modular. All of the ACE data query, etc. capabilities can be made available within an application module, without requiring that module to link in and integrate a large amount of foreign software.

Data Storage and Access - IAC data may be stored in an IAC workspace, an IAC database, and/or the host file system. Organization and use of these storage areas are detailed in Section 5.

For most applications, effective IAC operation requires a generous amount of real or virtual memory, to be used for user-specific workspace(s). A workspace is part of the ACE executive, and some amount of workspace is needed by any module which calls IAC utilities. The workspace provides a highly dynamic storage capability, and its use allows the elimination of arbitrary fixed size storage limitations.

An IAC database is organized in the form of a host directory or subdirectory. The IAC system provides a database catalog (file spec IACCAT.IAC) which contains characteristics of each IAC file stored in the database; the catalog is used to access files, and may be queried to locate and describe files of

interest. IAC also provides a database activities file (file spec IACACT.ACT) which manages the database for multi-user concurrent access.

Each IAC file consists of a data file and/or an associated narrative text file. An IAC file may be stored in a database, in a workspace, or in an arbitrary host directory. (In the case of the workspace, the text file is usually stored in the default directory, which for this purpose is considered to be a logical extension of the workspace).

An IAC file is identified by a four part spec, of the form

```
name:number.type;version
```

where ':number' and ';version' are optional inputs.

An IAC data file may be of class RELATION, ARRAY, USER, OR ANY. The first three classes are considered to be 'structured' files, which allow various degrees of IAC manipulation and query; class ANY is considered to be 'unstructured'. The data file may in some cases be stored simply as a host file on a host directory, independent of any database or workspace.

The host file system is used as a mechanism for providing editing and other host operating system capabilities. The host file system also is important in supporting the operation of the IAC database, since the various component files within the database are physically stored as separate host files.

2.0 EXECUTIVE COMMANDS

The IAC executive program is called ACE (Analysis Capability Executive). You control the sequence of tasks within ACE by giving either complete or partial commands, and by relying upon the command processor to prompt for any missing information. In other words, the user/executive interface provides you with a dual command language and tutorial prompting capability. Thus the executive can 'adapt' to various modes of input, consistent with your level of experience in using each particular command.

As a simple example, the following four ACE command sequences are equivalent.

```
ACE >MERGE RELATION NODES.DAT/101 NODES.NEW

ACE >MERGE RELATION NODES.DAT+
/101 NODES.NEW

ACE >MERGE RELATION NODES.DAT
/start-index-value >/101
MERGE-rel-spec >NODES.NEW

ACE >MERGE RELATION
rel-spec >NODES.DAT
/start-index-value >/101 NODES.NEW
```

In order to support the dual command/tutorial interface, ACE defines all commands as a series of 'nodes' (parts and sub-parts), and stores the definitions in a table type format. For each node the table defines an appropriate user prompt, user aids in the form of help and examples, permissible data input classes and types, and relationships between the given node and any lower level nodes. The command processor compares your input against the structure of the table, and transfers from node to node within the table as required. Whenever additional input is needed, the prompt associated with the corresponding node is displayed to you. The processor provides appropriate node position transfers and recovery in case of input errors.

The user/executive interface also provides the capability to predefine and then reference an auxiliary sequence of command or data items. This may be done via two methods. The first is to simply place the auxiliary input on a

host file, and then to reference this file during execution. The second method is to use the ACE macro and amacro substitution capabilities (see Section 2.2). It may also be useful to combine these methods, e.g. for implementing analysis solution path sequences, where specific file names etc. need to be inserted at run time.

Host system identified execution errors (e.g. arithmetic overflow) can occur either during executive computations or within application modules being run by the executive. The IAC philosophy is to trap all such errors, report them and return control to the interactive user within the executive program for a decision on how to proceed.

Output from ACE may be placed in one or more files, depending upon your specifications. Output includes the major computational and query type output, the tutorial prompts, help information, messages, and any module interface associated output.

2.1 COMMAND DIAGRAMS

The ACE commands are described with the aid of the command diagrams of Figure 2.1-1. It is suggested that initially you merely scan the top level boxes in these diagrams, and review the facing page text information, in order to develop a basic understanding of the available commands. As you become more experienced, you may find it helpful to understand and reference the diagrams in detail; this will provide information on required versus optional items, valid item types, relationships between groups of items, etc. The following notes are pertinent to a more detailed usage of the command diagrams.

1. You may provide input to ACE via either a command mode, a tutorial prompt mode, or a selected combination of the two modes.

2. Each box within a command diagram represents a syntactic command 'node'. The box encloses the node-associated user prompt. At any node you may input data for that node and following nodes. User help and examples information is available at most command nodes.

3. Above each box is given the node 'class' and sometimes a 'type'. A prefix "-" on the class indicates an optional node. Node classes and types are defined in Table 2.1-1.

4. Each node box indicates a single occurrence, unless the enclosed prompt is followed by an item in parentheses. A number item indicates the fixed number of occurrences, and an "*" item indicates a variable (one or more) number of occurrences.

5. Dash-line boxes indicate that the nodes are only typical examples in a general command structure.

6. A node-associated user input item is called a 'token'. Node classes having directly associated tokens are KEY, NUL, SYS, SYM and VAL.

7. Input tokens for node classes KEY and SYS may be abbreviated to leading characters which are unique within the particular usage context.

8. An alphanumeric token is a sequence of alpha and/or numeric characters. Alpha includes "A".."Z" and "_", while numeric includes "0".."9".

9. The six class names beginning with "G" indicate 'group' nodes. A group node represents all of the associated lower level nodes.

10. A user prompt beginning with the 5 characters "list(" indicates that the next level group members must either be collectively enclosed within parentheses or/and be separated from one another by commas.

11. 'Control inputs' may be used at any node in order to display associated help and examples information, to transfer from node to node, etc. A control input consists of a "%" followed by other appropriate characters. For example, "%H" displays help information at the current node, and "%UU" may be used to move two node levels upward.

System Type Tokens - IAC system type tokens begin with a dollar sign "$", and are used for many important purposes. The following is a list of common system tokens with their corresponding definitions.

$S	Same as last previously input node of this kind
$I	Index of relation, or all indexes of array
$Ii	Index i of array
$A	All attributes in array or relation
$Ai	All attributes associated with index i of array
$L	Length of a character attribute value
$D	Dimension (number of indexes) within nonscalar attribute
$C	Cardinality (number of elements) within nonscalar attribute
$Ci	Cardinality of index i within nonscalar attribute
$REAL	Real part of complex attribute
$IMAG	Imaginary part of complex attribute
$MAG	Magnitude of complex attribute
$ANG	Angle (radians) of complex attribute

Some Helpful Hints - All of your command input to ACE (except for general strings) must use upper-case characters.

You may use a plus sign "+" as the last non-blank character on an input line to continue the line.

You may abbreviate keywords and system tokens to leading characters which are unique within the usage context. For example, "DEF A" is equivalent to "DEFINE ARRAY", and "$R" to "$REAL".

The ACE commands are designed to be very versatile and powerful, and they provide a number of options which you may use only infrequently. If you are operating in command input node, you can simply skip over such options. If you are operating in tutorial prompt mode and you see a prompt which you don't want or don't understand, simply hit the carriage return to reject it.

If at a particular node you want to find out what your other options are, type in a bad input (e.g. "$XX" is always illegal). ACE will respond with a message, and list all the various correct options. Note that at the top level (i.e. the "ACE >" prompt), this is an easy way to get a listing of all the available ACE commands.

If you have made some serious mistakes in a command, or you are confused and want to start over at the top level, you may enter the two characters "%K" immediately after any prompt. This will kill the command and return you to the top level "ACE >" prompt.

You will often want to obtain help and examples associated with an overall command. Remember that, once you have given the command keyword (e.g. JOIN), you are positioned at a lower-level node where upper-level help is not available. To get what you want, give the keyword and a carriage return, and then type "%UHE". This moves you up one level to the appropriate position, and then displays help and examples, e.g.

```
ACE >JOIN
rel-spec1 >%UHE
```

A carriage return <CR> can be used to reject an optional input. For example

```
print-qualifiers ><CR>
```

rejects the optional print-qualifiers item and brings up the prompt for the next item. If the input item is required, the prompt will be repeated.

Some input items have associated "auto-inputs"; these are provided for you automatically by ACE, so that you won't have to type them in. For example, the left parenthesis in

```
list(att-names) >(
```

is an auto-input. To reject an optional input item and associated auto-input, simply hit <CR>. To accept the auto-input and move on, enter additional input or simply a space. For example,

```
list (att-names) >(<CR>
```

rejects the parenthesis, and

```
list (att-names) >(<space-bar><CR>
```

accepts it and moves on to the next prompt.

Some Basic Terms - The term "data structure" is a fancy designation for "file", but it implies that the file is organized in some standard manner. A data structure may be a "relation" (table) or an "array".

"Attribute" means a variable within a data structure, e.g. a column within a relation, or a matrix or label within an array.

"Tuple" means a logically associated group of attribute values, e.g. a row of values within a relation or an associated group of matrix/label values within an array.

"Spec" indicates a specification. This usually refers to a host file, or to an IAC relation or array.

Class	Type	Description
KEY	-	Keyvalue (predefined character sequence, without quotes)
NUL	-	Null token (character "*")
SYS	-	IAC system token (initial character "$")
SYM	-	Special IAC symbol: ( ) [ ] , / = .. && //
GOR	-	Group of ordered nodes
GUN	-	Group of unordered nodes
GAL	-	Group of alternate nodes
GOS	-	Group of ordered nodes with separators
GUS	-	Group of unordered nodes with separators
GAS	-	Group of alternate nodes with separators
EXT	-	External (e.g. callable routine) defined node
VAL	I1	Integer value (1 numeric storage unit)
VAL	R1	Real value (1 numeric storage unit)
VAL	R2	Real value (2 numeric storage units)
VAL	Z2	Complex value (2 numeric storage units)
VAL	Z4	Complex value (4 numeric storage units)
VAL	L1	Logical value (1 numeric storage unit)
VAL	Cn	Character value (n character storage units)
VAL	C*	Character value (variable character storage units)
VAL	CFILE	IAC (cataloged) file spec
VAL	HFILE	Complete host file spec, optionally within quotes
VAL	NAME	Maximum 10-character alphanumeric, starting with alpha
VAL	NUMBER	Maximum 10-character integer
VAL	NAMNUM	NAME, followed by optional ":" and NUMBER
VAL	TYPE	One of the values I1, R1, R2, Z2, Z4, L1, Cn, C*
VAL	STRING	Character value, within quotes
VAL	AR	Arithmetic value (readable via single FORTRAN F format)
VAL	NA	Nonarithmetic value, possibly within quotes

Table 2.1-1: Command Node Classes and Types

The top-level node in the ACE executive command language has the prompt-name ACE. This prompt name is displayed as shown whenever ACE is ready for a new command.

```
ACE >
```

You may then respond with any ACE command. If the command is complete, it will be processed; if not, ACE prompts for the next missing node (i.e. command diagram box). Typical sequences are shown for a complete command and a partial command.

```
ACE >SET IO JOURNAL=SAVE.MY_WORK
*INFORMATION* Last ACE command processed
ACE >

ACE >GETS X.DAT
new-file-spec >
```

A complete command consists of a statement followed by an optional statement-disposition. The disposition $EXECUTE (default) causes the command to be executed, while $NOEXECUTE only causes the command to be checked for syntax. Since the disposition is optional and is not followed by any required nodes, it is usually not prompted for.

```
ACE >TALLY BIG.TABLE X GT 0 /$NO
*INFORMATION* Last ACE command processed
ACE >

ACE >TALLY BIG.TABLE X GT 0
73050 TUPLE(S) satisfy the conditions.
*INFORMATION* Last ACE command processed
ACE >
```

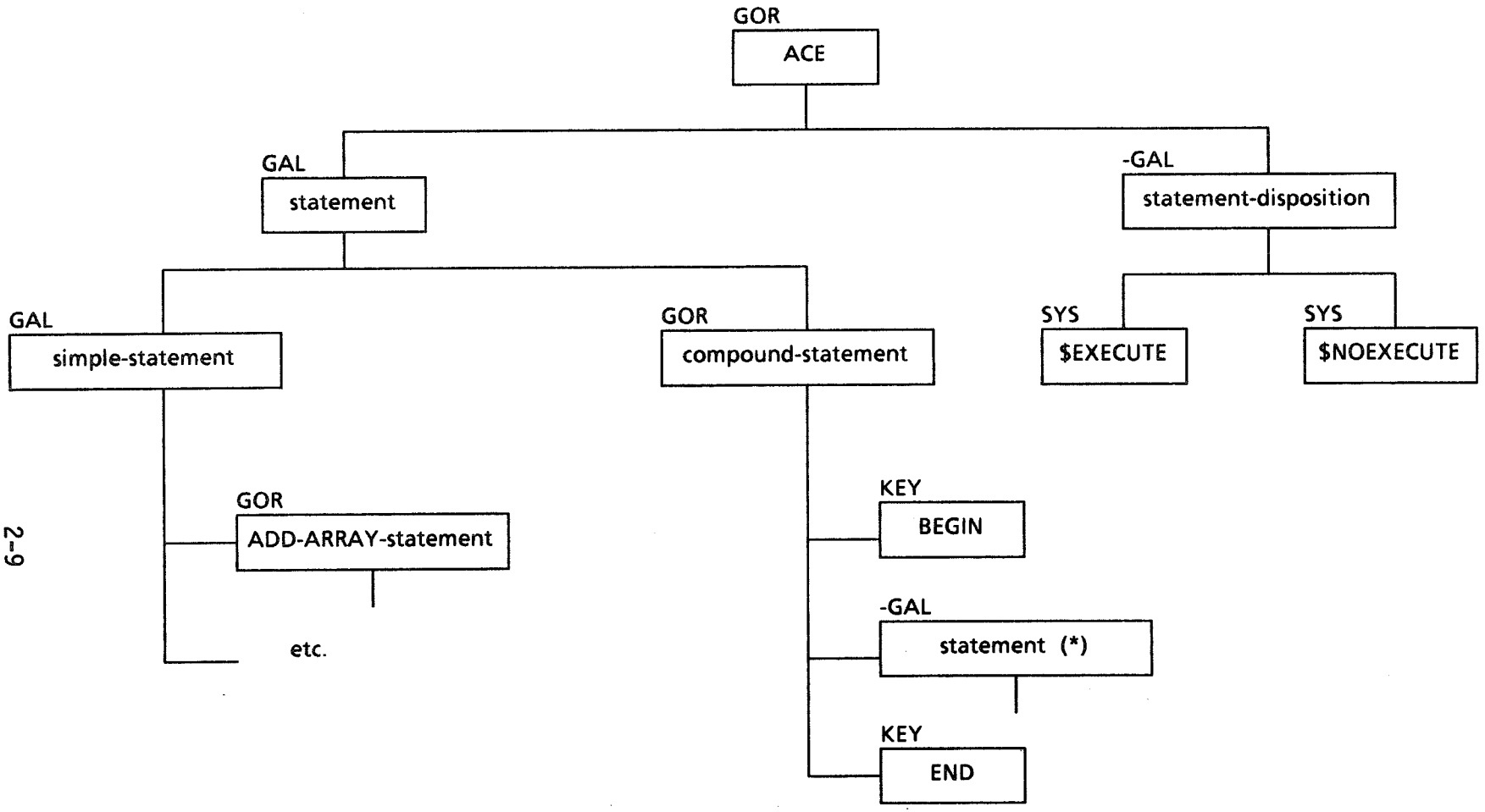
Your command may be a compound statement; i.e. BEGIN, an arbitrary number of statements, and END. All the included statements are executed together. You might use the compound statement if you need to temporarily leave your terminal, and want execution to continue. Note that a plus sign at the end of any input record causes continuation of the following record.

```
ACE >BEGIN DELETE/W *.*;* +
GET NODES.DAT +
SHOW FILE NODES.DAT +
SET IO OUT=SAVE.DAT +
PRINT NODES.DAT NODE,X,Y * +
SET IO OUT='*' +
END
```

You may often want to obtain help and/or examples for a particular overall command. Give the command keyword, followed by a control request %UHE (moves up one level and then displays help and examples). Note that if you give incorrect input at any node, a menu of correct input possibilities will be displayed for you.

```
ACE >HISTOGRAM
HISTOGRAM-qualifiers >%UHE

ACE >XXX
*ERROR* The token at the end of:
        XXX
        did not satisfy . . .
      .
      .
      .
```

statement statement-disposition



ADD ARRAY adds two arithmetic arrays. A coefficient times the "add-array" is added to another coefficient times the "array", and the result is stored in the "array". The two arrays and their coefficients must all be of the same type (I1, R1, R2, Z2 or Z4).

```
ACE >ADD ARRAY A.DAT B.DAT
ACE >ADD A 1.5 A.DAT -2.0 B.DAT
ACE >ADD A A.DAT/51,11 B.DAT
```

The "array" node consists of an optional coefficient (default is unity), an array spec, and an optional attribute reference (the attribute reference is required if there are more than one attribute in the array core, i.e. associated with index 0). The "add-array" node is identical in form to the "array" node.

```
array >A.DAT
array >1.5 A.DAT A1
array >[1.0, 2.0] Z.DAT
```

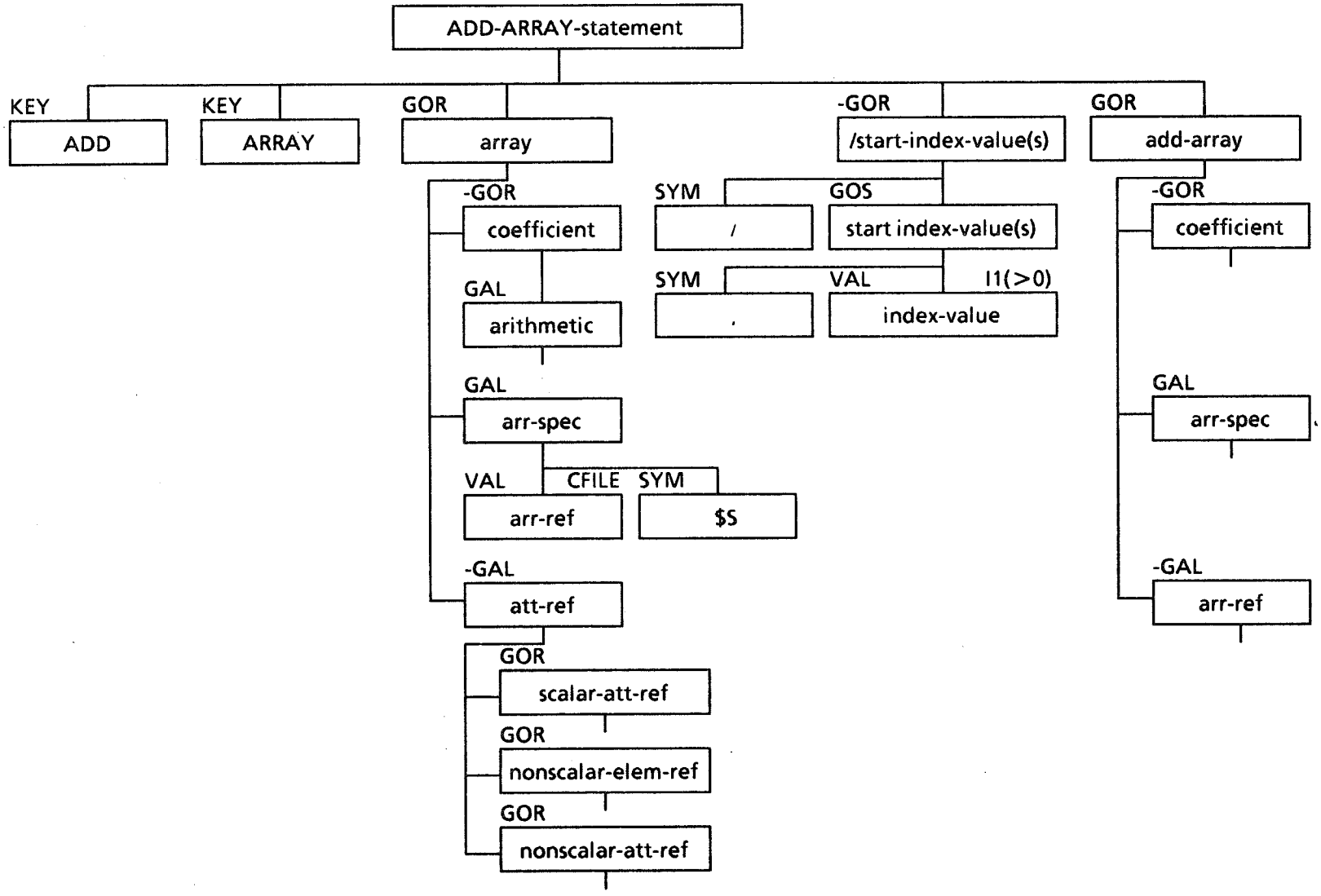
The "/start-index-value" specifies an optional starting index position within the array, at which the first value in the add-array is to be aligned prior to adding. The starting index position may be given either via a single index 0 value, or as a multiple index value; default value is 1. If the add-array after alignment extends beyond the boundaries of the array, the add operation is performed but a warning message is written.

```
/start-index-value >/101
/start-index-value >/11,21
/start-index-value >/5,17,1
```

The "att-ref" node may reference a scalar or nonscalar attribute within the array index 0. If a subset (partition) of a nonscalar attribute is referenced, only the referenced partitions are added; partitions in the two arrays must correspond in size.

```
att-ref >A
att-ref >B[3]
att-ref >B[2..3, 4..6]
ACE >ADD ARRAY A.DAT A B.DAT B[3]
```

ADD ARRAY array /start-index-value(s) add-array



ADD-ARRAY-statement

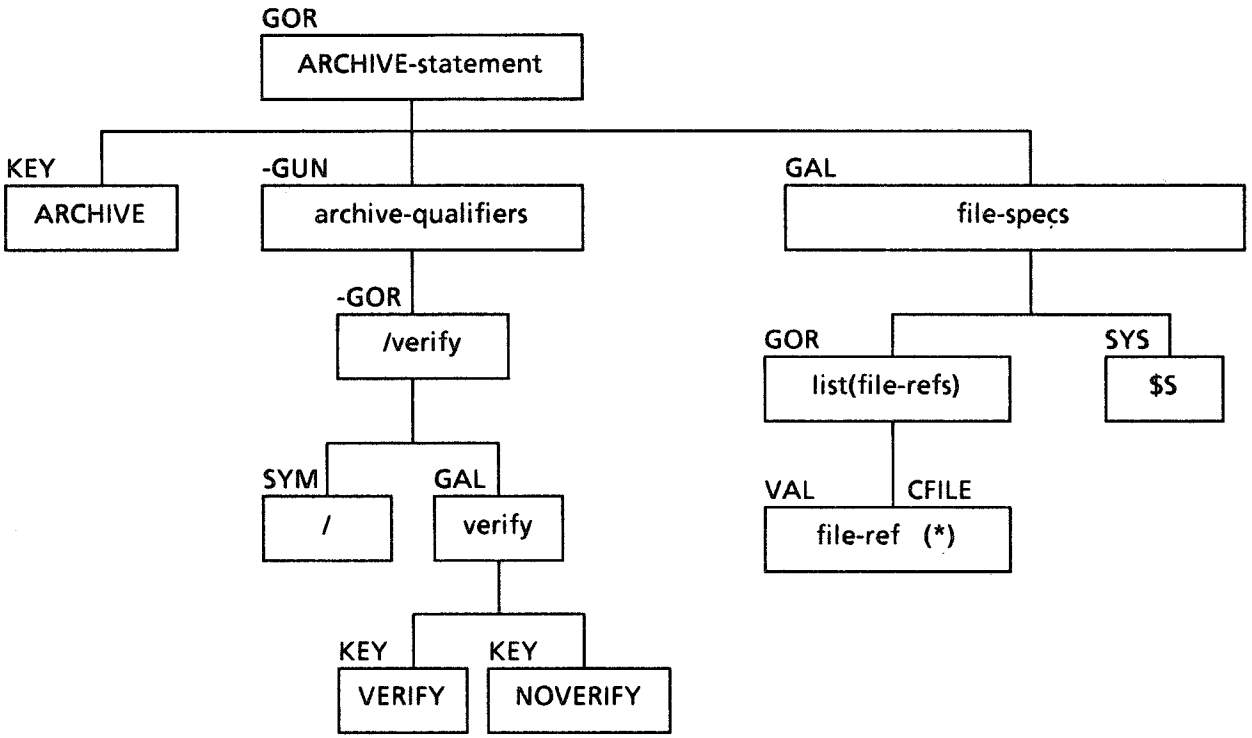
ARCHIVE allows you to archive files from an IAC database. The verify option (default is NOVERIFY) lets you confirm the archiving.

The database archiving concept is very simple, and relies upon the user to select the archiving medium and location. The ARCHIVE command simply results in the status flag in the IAC database catalog being set to A (meaning archived). All other file attributes stored in the catalog remain available for query; however, the archived file cannot be retrieved or updated via the normal IAC provided tools. It is your responsibility to dispose of the IAC data and/or text file, and to restore these files prior to using the UNARCHIVE command.

Following the ARCHIVE command, you may want to use LIST/D to list files in the database, GETC to obtain the database catalog for query of overall file attributes, or eventually UNARCHIVE to restore the file or DELETE/D to delete the file.


```
ACE >ARCHIVE A.DAT

ACE >ARCHIVE A.DAT,B.DAT,*.LIS

ACE >ARCHIVE/V X.Y
```

ARCHIVE archive-qualifiers file-specs



ARCHIVE-statement

CATALOG enters an IAC file description into the database catalog. The description consists of a set of file attributes, whose values you may specify and/or allow to be defaulted. You must have the associated host files (data file and/or text file) in the database directory prior to cataloging. The verify option (default NOVERIFY) lets you confirm results of the cataloging.

You may specify the following file attributes.

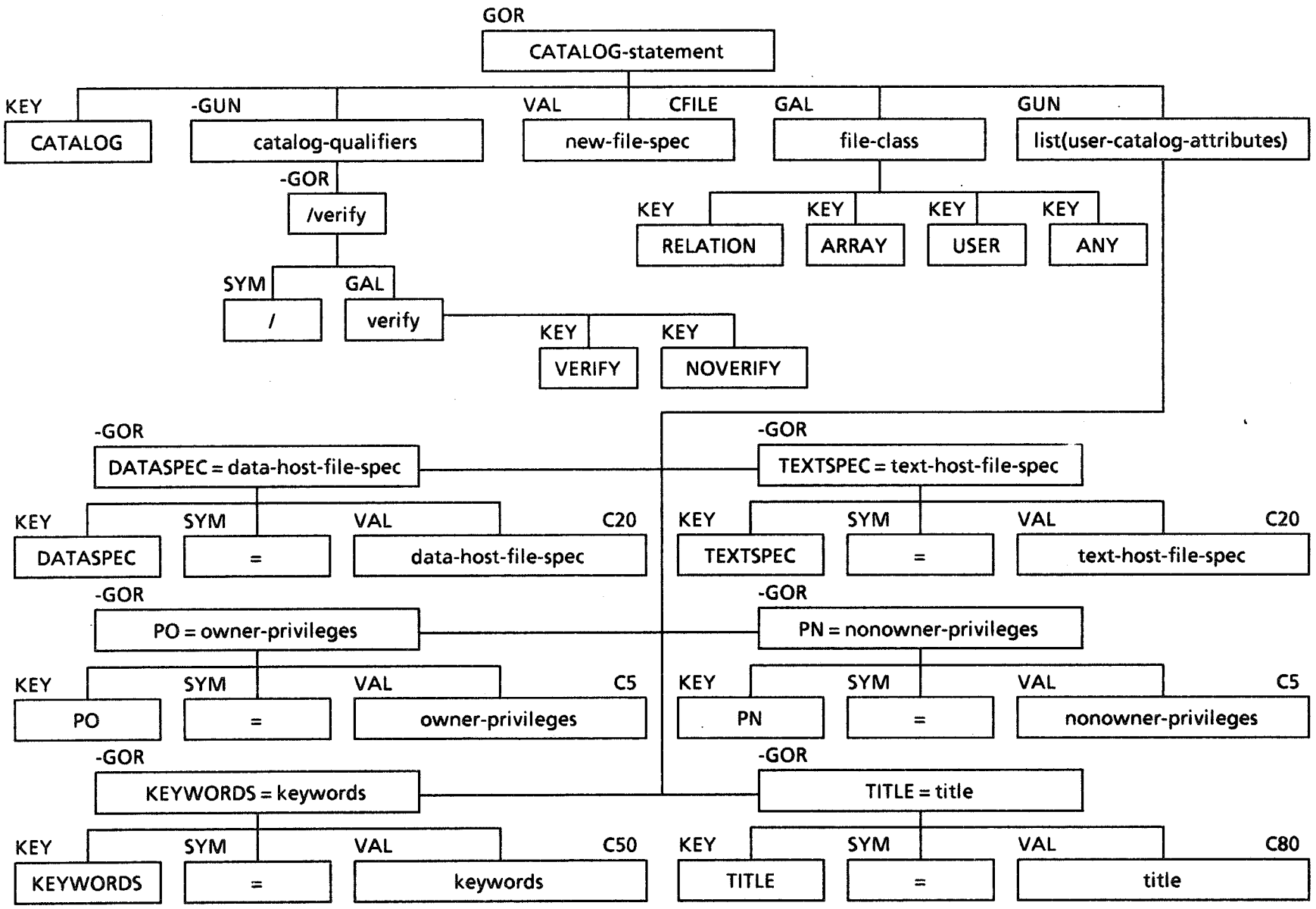
```
DATASPEC=data-host-file-spec  (default=' ')
TEXTSPEC=text-host-file-spec  (default=' ')
PO=owner-privileges           (default='*')
PN=nonowner-privileges        (default='*')
KEYWORDS=keywords             (default=' ')
TITLE=title                   (default=' ')
```

In general a blank file attribute value indicates absence of data for the attribute (e.g. blank value for PN indicates no privileges for the nonowner). An asterisk value indicates that a database dependent default value is to be generated for insertion into the database catalog.

Following the CATALOG command, you may want to use LIST/D to list files in the database, or GETC to obtain the database catalog for more detailed query.


```
ACE >CATALOG A.REL RELATION D=A.DAT

ACE >CATALOG X:2.MAT ARRAY (D=X2.MAT,TE=T2.X)

ACE >CATALOG/V A.REL REL D=A.DAT


ACE >CAT COMP.LIS ANY
list(user-catalog-attributes) >(D=COMP.LIS,+
TEXT=COMP.TXT,PN=R,KEY='MASS TRUSS JULY84',+
TITLE='DESIGN 2 DYNAMIC TIME RESPONSE')

ACE >CAT PROB:3.ARR ARRAY+
(D=PROB3.ARR, TITLE='TEST CASE MAY 9')


nonowner-privileges >RWDAL

nonowner-privileges >R

nonowner-privileges >'*'

nonowner-privileges >' '
```

CATALOG catalog-qualifiers new-file-spec file-class list (user-catalog-attributes)



CATALOG-statement

CHANGE alters values in a relation or array. Changes are accomplished via a list of "change equations", of the general form "old-value-reference = new-value".

The "old-value-ref" node may involve a scalar or a nonscalar attribute.

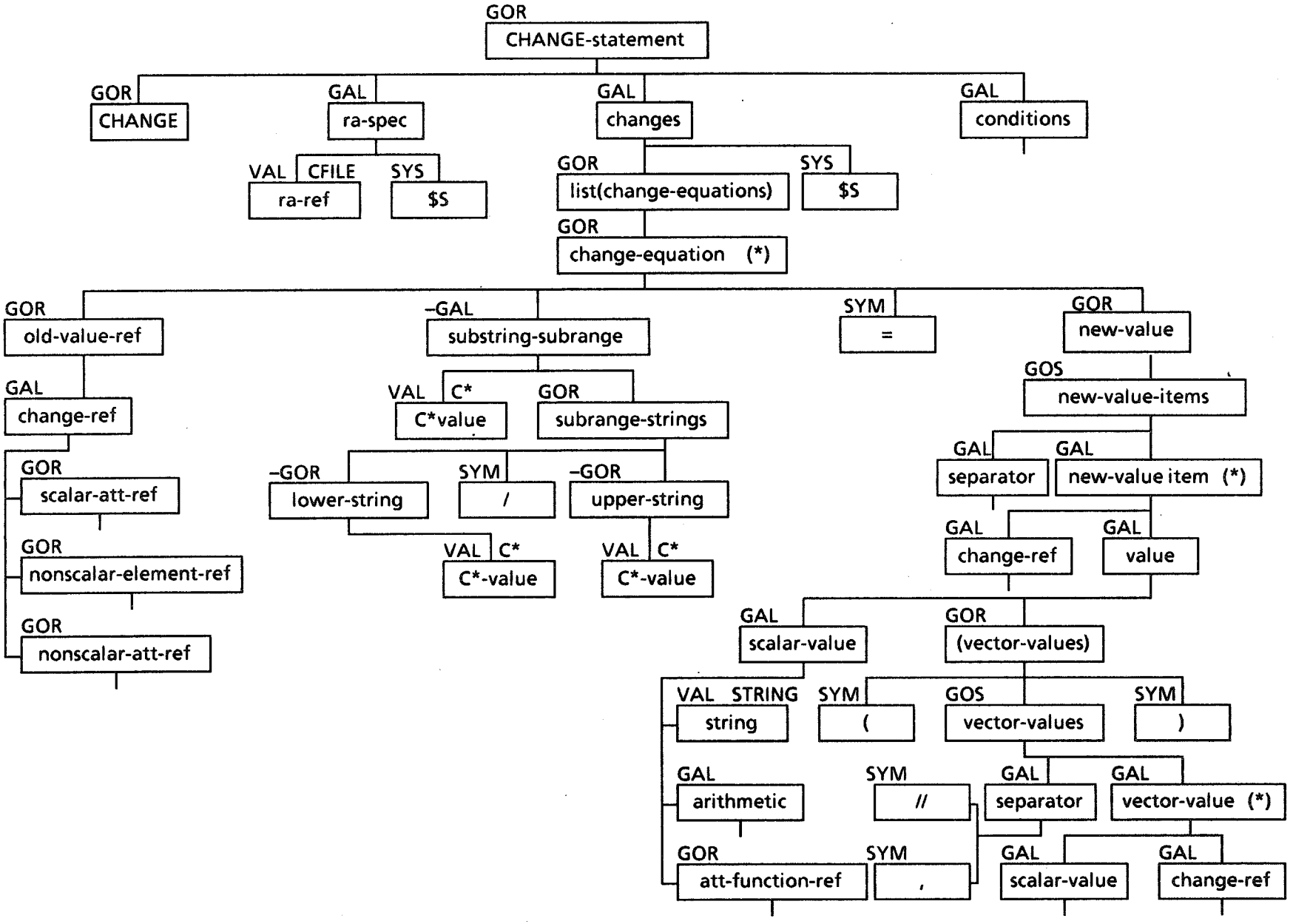
For a character type attribute reference, a substring may be changed. This is accomplished by giving substring index positions and/or subrange strings. In the first example, only positions 41 through 80 in the attribute TXT are selected for change. In the second example, the substring to be changed is further restricted to only the characters from the beginning of the string AB to the end of the string XY. In the third example, the substring is not limited by index positions, but only by the lower and upper substring subranges.

The "new-value" node may consist of one or more new value items. Multiple items are catenated (separator "//" denotes string catenation, and "," denotes vector catenation).

Note - Multiple change equations and vector-value changes are not currently implemented.

```
ACE >CHANGE AB.DAT L1=2 L1 EQ 1

ACE >CHANGE X.Y (X=0.  Y=0.) Z GT 1

ACE > CHANGE A.TXT CH=XYZ  CH EQ ABC

ACE >CHANGE A.DAT S[3]=1.5 X GTN Y
old-value-ref >S[2..3]=(2.5,3.5) s[1] EQ 0

old-value-ref >TXT[41/80]

old-value-ref >TXT[41/80] AB/XY

old-value-ref >TXT AB/XY

new-value >1.0

new-value >(1.0,2.0),(X[3..5]),(7.0)

new-value >'XYZ'

new-value >'XYZ'//A[3/8]//A[1/2]
```

CHANGE ra-spec changes conditions



CHANGE-statement

CLEAR IO re-initializes all ACE IO file specifications to their standard values. The IO files and their respective standard values are as follows.

INPUT = '*' indicates a return to the current position within the primary input file.

AMACRO = '*' indicates a return to the current position within the primary input file.

OUTPUT = '*' indicates the interactive terminal or batch log file.

PROMPT = '*' indicates the interactive terminal or batch log file.

MESSAGE = '*' indicates the interactive terminal or batch log file.

ECHO = ' ' indicates absence of the echo operation.

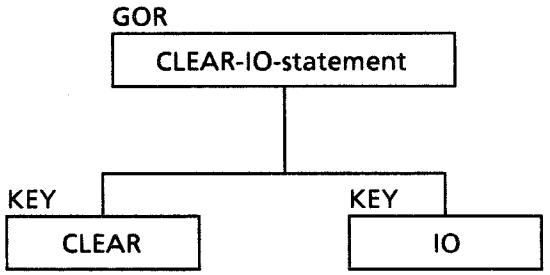
JOURNAL = ' ' indicates absence of the journal operation.

Note that if the primary input file is the interactive terminal, "a return to the current position within the primary input file" simply means a positioning to the current input line from the terminal.

IO file specifications may be set via the SET IO command, and displayed via the SHOW IO command.

ACE >CLEAR IO

CLEAR IO



CLEAR-IO-statement

ACE >CLEAR MACRO

CLEAR MACRO re-initializes all ACE MACRO parameters to their standard values. The parameters and their respective standard values are as follows.

STATUS = ON indicates that macros are active, i.e. that the occurrence of a single "@" symbol is to be interpreted as the start of a macro reference

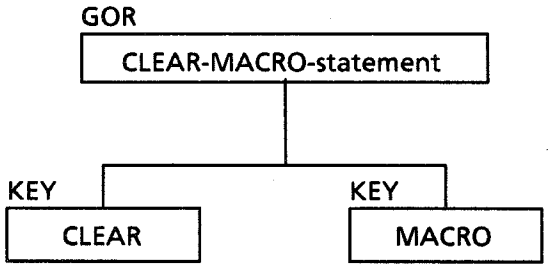
RELATION = MACRO.RELATION indicates the spec of the default macro relation in the workspace is to be interpreted as the start of a macro reference.

ID = ID indicates the name of the default id attribute within the macro relation.

TEXT = TEXT indicates the name of the default text attribute within the macro relation.

MACRO parameters may be set via the SET MACRO command, and displayed via the SHOW MACRO command.

CLEAR MACRO



CLEAR-MACRO-statement

CLEAR MESSAGE re-initializes ACE message control parameters to their standard values. The parameters and their respective standard values are as follows.

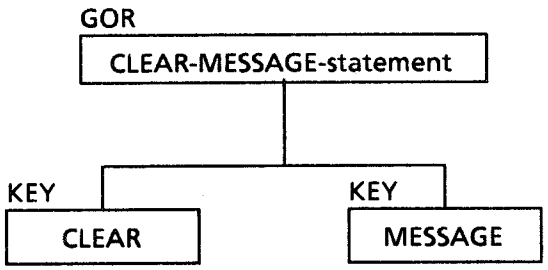
INFORMATION = ON indicates that ACE information level messages are to be displayed.

WARNING = ON indicates that ACE warning level messages are to be displayed.

Note that ACE error level messages are displayed in any case. Message control parameters may be set via the SET MESSAGE command, and displayed via the SHOW MESSAGE command.

ACE >CLEAR MESSAGE

CLEAR MESSAGE



CLEAR-MESSAGE-statement

COMPUTE computes summary information for selected values within a relation or array.

```
ACE >COMPUTE A.DAT (X,Y) *
```

Summary information may be obtained for the following compute options (default is all compute options), where n is the number of tuples selected by the conditions.

```
ACE >COMP/MEAN DEV A.B X,Y *
```

SUM	= sum of n values.
ASUM	= sum of n absolute values.
MAXIMUM	= maximum of n values.
MINIMUM	= minimum of n values.
RANGE	= maximum minus minimum.
PRODUCT	= product of n values.
MEAN	= arithmetic mean of n values.
GMEAN	= geometric mean of n values.
MEDIAN	= center value (at n/2 for n even).
MODE	= most often occurring of n values (first mode if mode is not unique).
RSS	= root sum square of n values.
DEVIATION	= standard deviation of n values = (sum of n ((value-mean)**2))**(1/2))
CHI2	= CHI-squared = sum of m ((fo-fe) **2/fe) over n values; here m = number of intervals; and fe, fo = expected, observed frequency.

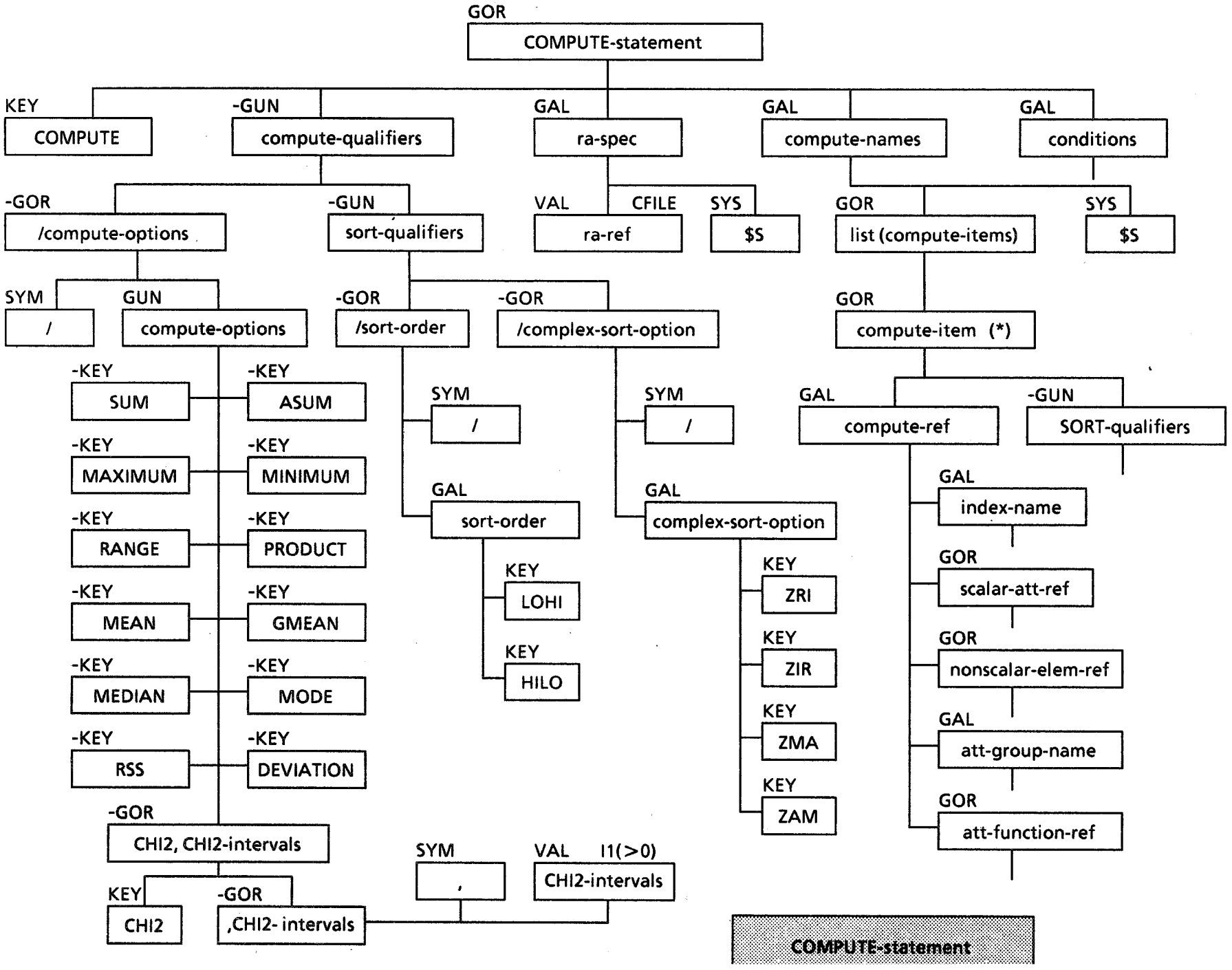
The sort-qualifiers influence order of the n values for computation (defaults are LOHI and ZRI). These qualifiers are infrequently used, but may affect results of MEDIAN, MODE, and CHI2.

```
ACE >COMP/HILO A.B X  X GT 0.
```

The compute names may include scalar attributes, nonscalar attribute elements, etc.

```
ACE >COMP A.B (X, STRESS[3]) *
```

COMPUTE compute-qualifiers ra-spec compute-names conditions



Conditions allow selection of "tuples" (i.e. rows in a relation or positions in an array) via an extended FORTRAN kind of syntax. The conditions may consist of various kinds of condition definitions expressions, the "*" indicating no conditions (all tuples selected), or "$S" indicating that conditions are to be the same as the previously used conditions.

```
conditions >X GTN Y AND A NE 0.
conditions >*
conditions >$S
```

A simple condition definition is one of the following (each is explained in detail by a succeeding chart). Only the first three are operational in IAC Level 1.

scalar-ref	scalar-relop	scalar-value
scalar-ref	scalar-relopn	scalar-ref
scalar-ref	scalar-setop	(scalar-set)
nonscalar-att-ref	nonscalar-relop	(nonscalar-value)
nonscalar-att-ref	nonscalar-relopn	nonscalar-att-ref
nonscalar-att-ref	nonscalar-setop	(nonscalar-set)
nonscalar-att-ref	nonscalar-scalar-setop	(scalar-set)

```
conditions >ANGLE LT 90.
conditions >STRESS[3] GTN SB
conditions >J IN (1, 5,51..60)
conditions >NDEX[3..4] EQ (7,8)
conditions >NDEX[3..4] NEN MDEX[1..2]
conditions >CHAR[..] CAV ((A,B),(X,Y))
conditions >N2[1..20] CSE (1, 5..9, 50)
```

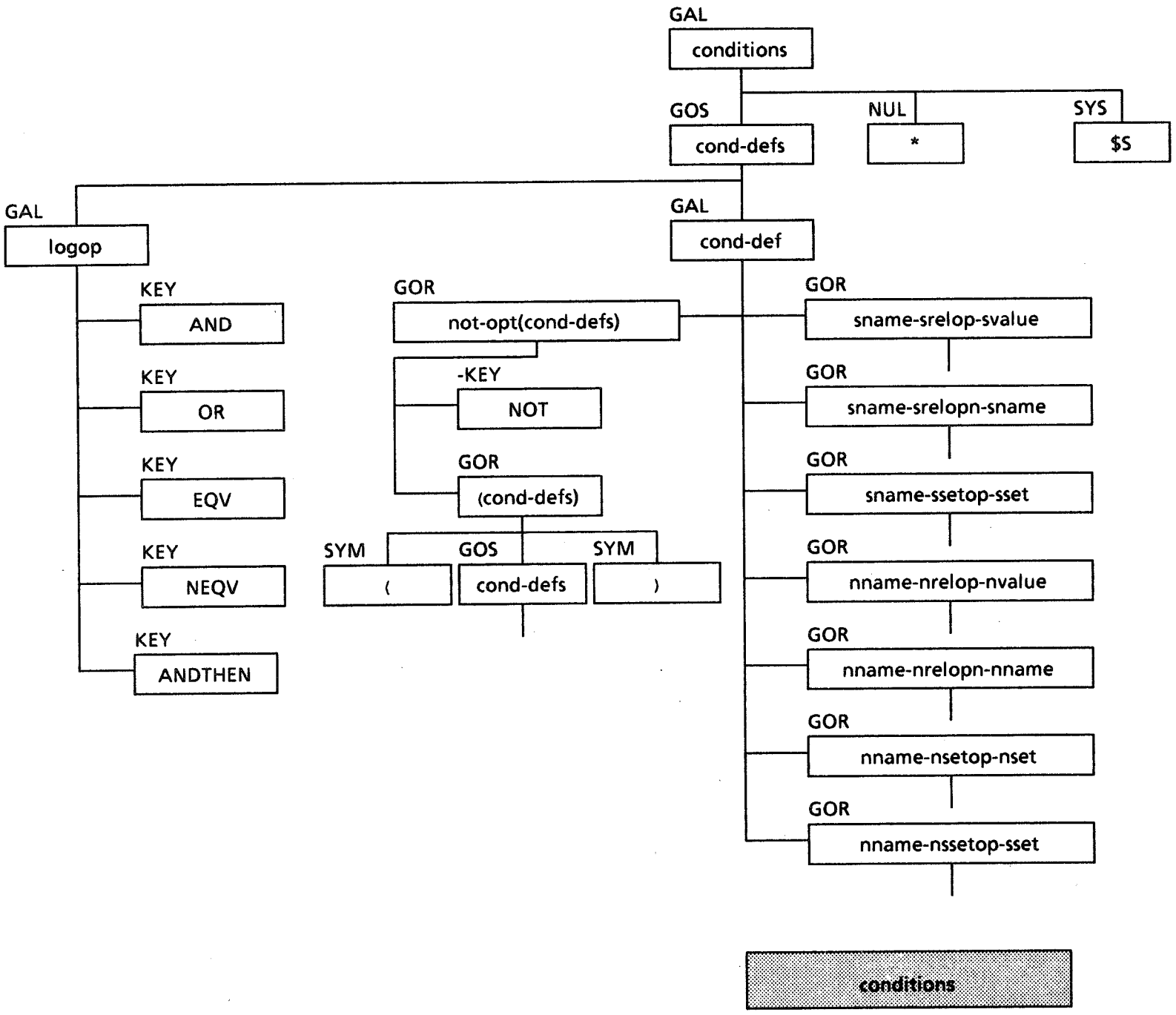
A condition definitions expression consists of one or more condition definitions, connected by one of the following logical operators.

AND	logical conjunction
OR	logical inclusive disjunction
EQV	logical equivalence
NEQV	logical nonequivalence (exclusive or)
ANDTHEN	logical conjunction with prior subset

```
conditions >X GTN Y AND A EQ 0.
conditions >X GEN Y OR AB LT 10.
conditions >F1 EQ 'N' EQV F2 EQ 'N'
conditions >X LT 5 ANDT Y EQ $MAX
```

A complex condition definition may utilize parentheses to determine order of evaluation. NOT prior to a parenthtized expression indicates logical negation.

```
conditions >(X GT 5 AND Y LT 0) OR Z GT 0
conditions >NOT(X GT 5 AND Y LT 0)
```


GAL
conditions
GOS
cond-defs
NUL
*
SYS
$S
GAL
logop
GAL
cond-def
KEY
AND
KEY
OR
KEY
EQV
KEY
NEQV
KEY
ANDTHEN
GOR
not-opt(cond-defs)
-KEY
NOT
GOR
(cond-defs)
SYM
(
GOS
cond-defs
SYM
)
GOR
sname-srelop-svalue
GOR
sname-srelopn-sname
GOR
sname-ssetop-sset
GOR
nname-nrelop-nvalue
GOR
nname-nrelopn-nname
GOR
nname-nsetop-nset
GOR
nname-nssetop-sset
conditions

The "sname-srelop-svalue" node is a condition involving a scalar named reference, a scalar relational operator, and a scalar value.

The available scalar operators are the following.

EQ	equal
NE	not equal
LT	less than
LE	less than or equal
GT	greater than
GE	greater than or equal
ME	most equal

Note that the ME operator selects the tuple containing the value most equal to the named reference. If there is more than one such tuple, they are all selected.

The scalar reference may be one of the following.

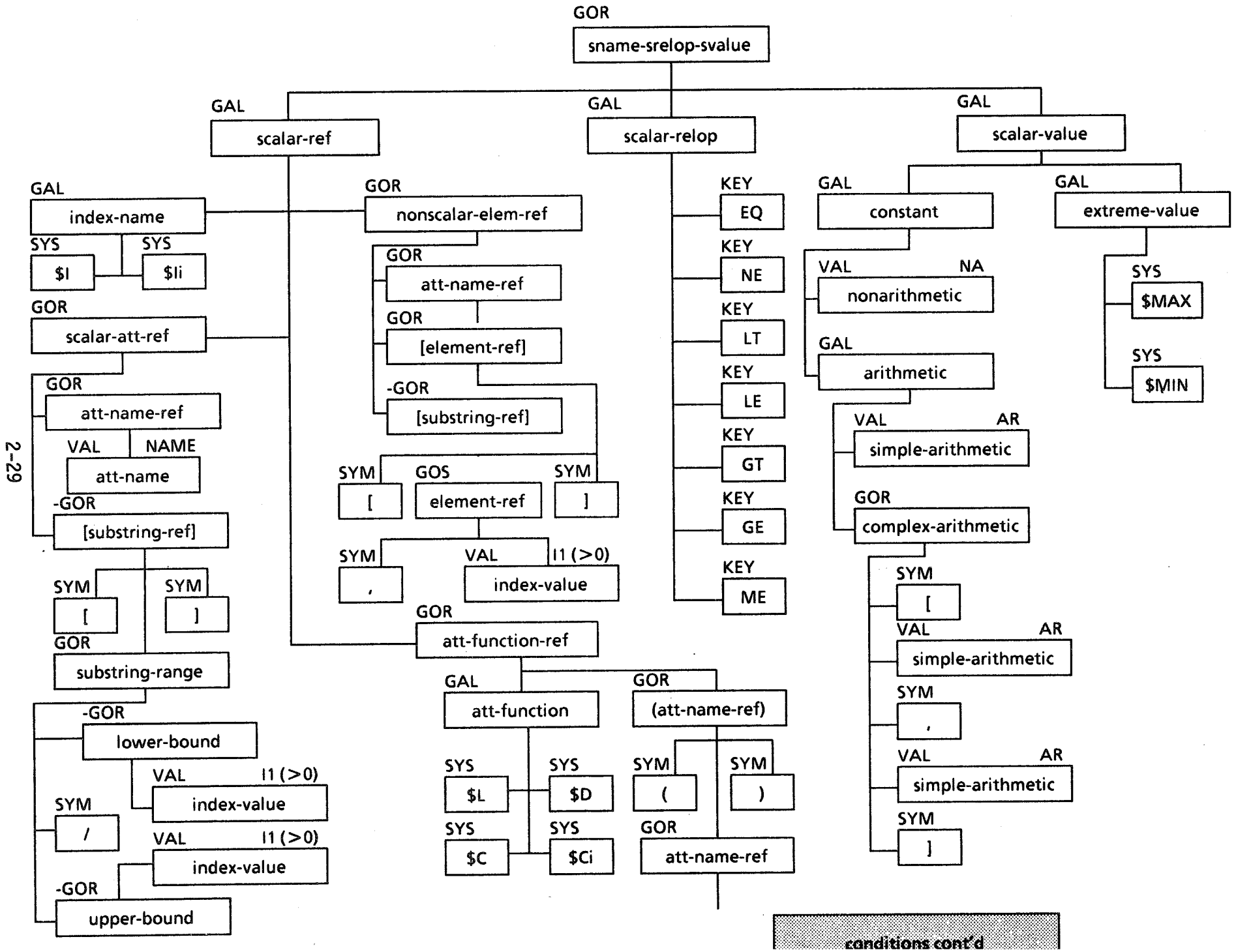
index name	e.g.	$I or $I0 or $I2
scalar attribute	e.g.	MX or TXT [41/80]
nonscalar element	e.g.	K[3] or TX[2] [21/30]
attribute function	e.g.	$L(A16) or $C2(MAT)

The scalar value may be one of the following.

simple arithmetic	e.g.	3 or 6.23 or 17.E2
complex arithmetic	e.g.	[1.5,2.E4] or [1.02,0.]
nonarithmetic	e.g.	XY or 'A,2' or .TRUE.
extreme value	i.e.	$MAX or $MIN

Note that a simple arithmetic is any value which can be read by a standard FORTRAN single "F" format. $MAX and $MIN denote the respective maximum and minimum values over all tuples. Note that character B is greater than A, C2 is greater than C10, logical .TRUE. is greater than .FALSE., and complex number sizes are compared based on modulus.

```
sname-srelop-svalue >$I LE 50
sname-srelop-svalue >X GT 90.
sname-srelop-svalue >TIME EQ $MAX
sname-srelop-svalue >ANGLE ME 90.
sname-srelop-svalue >ISTRING NE XYZ
sname-srelop-svalue >ISTRING EQ 'AB C'
sname-srelop-svalue >ISTRING[1/2] EQ AB
sname-srelop-svalue >ISTRING[4/] EQ C

sname-srelop-svalue >$I0 LE 25
sname-srelop-svalue >TEXT [5] [6/8] EQ 23Y
sname-srelop-svalue >$C2(MAT) EQ 3
sname-srelop-svalue >$L(TITLE) GT 80
sname-srelop-svalue >Y GT 17.1E2
sname-srelop-svalue >Z LE [1.5,2.E4]

sname-srelop-svalue >ALOG EQ .TRUE.
sname-srelop-svalue >CMPLX ME [1.,0.]
```


GOR
sname-srelop-svalue
GAL
scalar-ref
GAL
scalar-relop
GAL
scalar-value
GAL
index-name
SYS
$I
SYS
$Ii
GOR
scalar-att-ref
GOR
att-name-ref
VAL
NAME
att-name
-GOR
[substring-ref]
SYM
[
SYM
]
GOR
substring-range
-GOR
lower-bound
VAL
I1 (>0)
index-value
SYM
/
VAL
I1 (>0)
index-value
-GOR
upper-bound
GOR
nonscalar-elem-ref
GOR
att-name-ref
GOR
[element-ref]
-GOR
[substring-ref]
SYM
[
GOS
element-ref
SYM
]
SYM
,
VAL
I1 (>0)
index-value
GOR
att-function-ref
GAL
att-function
SYS
$L
SYS
$D
SYS
$C
SYS
$Ci
GOR
(att-name-ref)
SYM
(
SYM
)
GOR
att-name-ref
KEY
EQ
KEY
NE
KEY
LT
KEY
LE
KEY
GT
KEY
GE
KEY
ME
GAL
constant
VAL
NA
nonarithmetic
GAL
arithmetic
VAL
AR
simple-arithmetic
GOR
complex-arithmetic
SYM
[
VAL
AR
simple-arithmetic
SYM
,
VAL
AR
simple-arithmetic
SYM
]
GAL
extreme-value
SYS
$MAX
SYS
$MIN
conditions cont'd

The "sname-srelopn-sname" node is a condition involving a primary scalar named reference, a scalar relational operator, and a secondary scalar named reference.

The available scalar operators are the following.

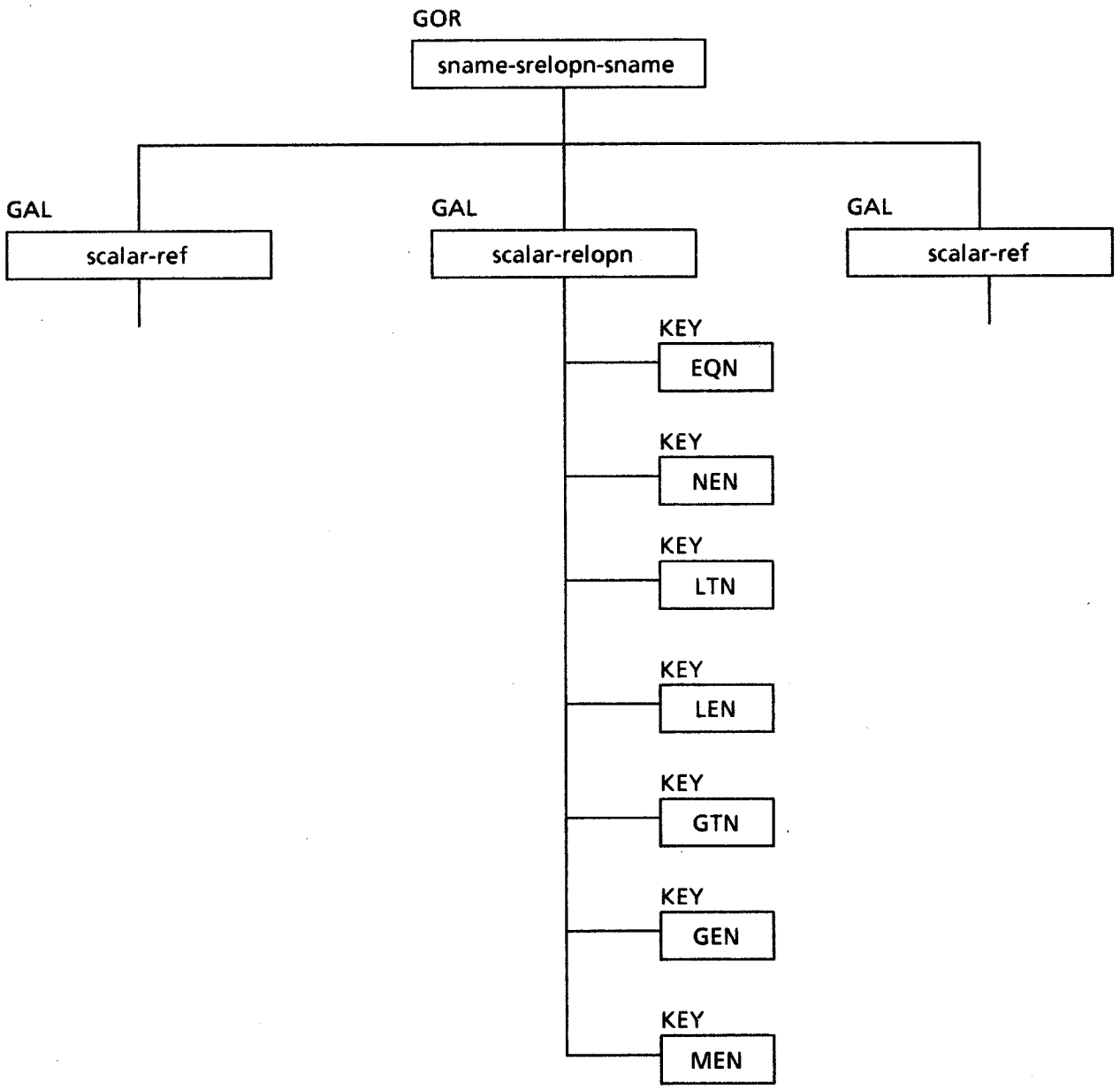
EQN	equal name
NEN	not equal name
LTN	less than name
LEN	less than or equal name
GTN	greater than name
GEN	greater than or equal name
MEN	most equal name

Note that the MEN operator selects the tuple containing the secondary named reference most equal to the primary named reference. If there is more than one such tuple, they are all selected.

The scalar reference may be one of the following.

index name	e.g.	$I or $I0 or $I2
scalar attribute	e.g.	MX or TXT[41/80]
nonscalar element	e.g.	K[3] or TXT[2][21/30]
attribute function	e.g.	$L(A16) or $C2(MAT)


```
sname-srelopn-sname >X GTN Y

sname-srelopn-sname >FLG[3]EQN FLG[4]

sname-srelopn-sname >$I1 EQN $I2

sname-srelopn-sname >KEY[1/5] NEN KEY[6/10]

sname-srelopn-sname >TEXT [3] [1/20] +
EQN TEXT [2] [1/20]

sname-srelopn-sname >$$C1(MAT) EQN $C2(MAT)

sname-srelopn-sname >X MEN Y
```



conditions cont'd

The "sname-ssetop-sset" node is a condition involving a scalar named reference, a scalar set relational operator, and a scalar set.

The available scalar set operators are the following.

IN	in the set
NI	not in the set
CAS	contains all of the strings
CSS	contains some of the strings
CNS	contains none of the strings

Note that IN holds if the named reference is equal to one or more of the values in the set; CSS holds if the named reference contains one or more of the substrings in the set.

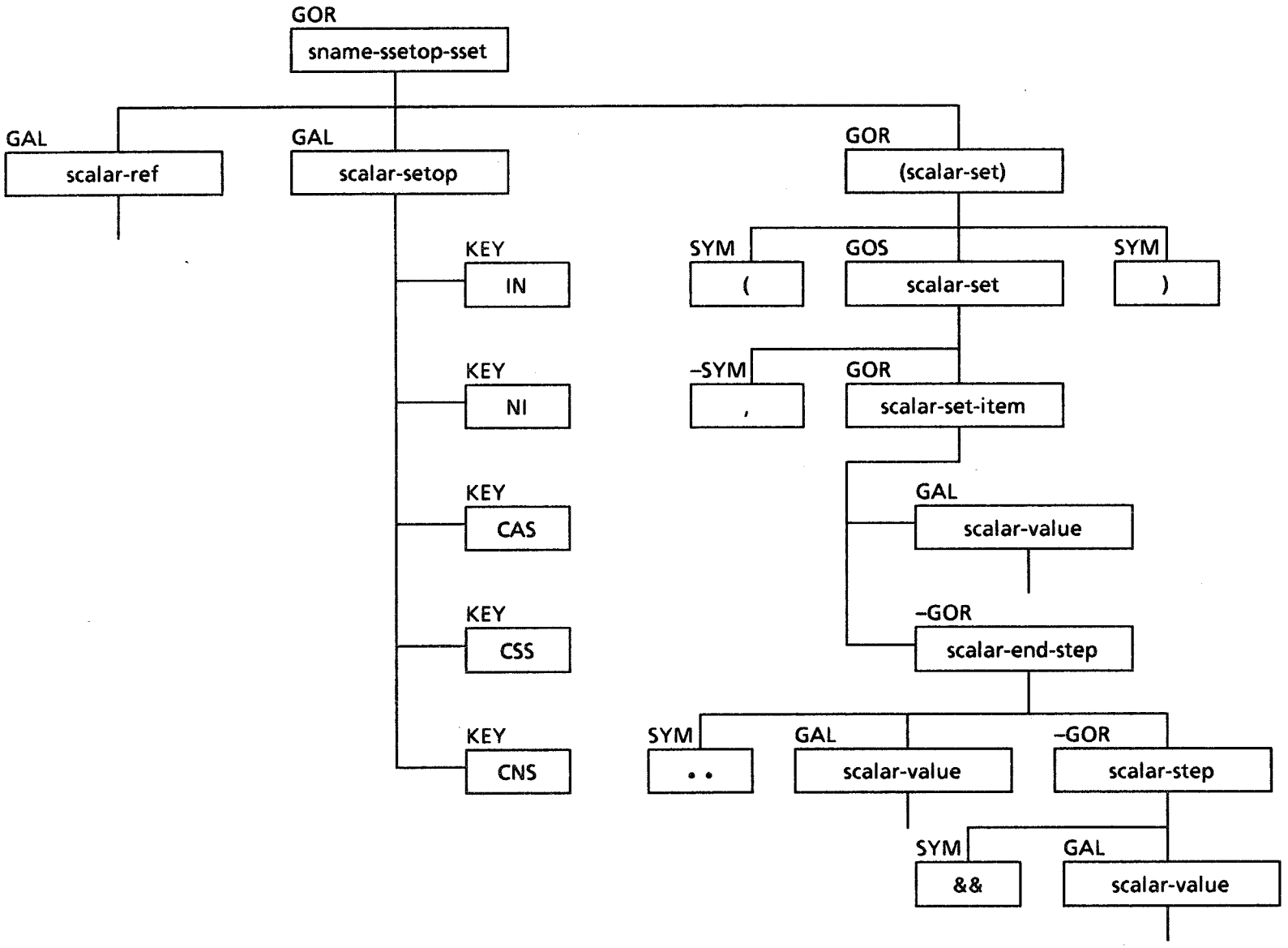
The scalar reference may be one of the following.

index name	e.g.	$I or Io or $I2
scalar attribute	e.g.	MX or TXT [41/80]
nonscalar element	e.g.	K[3] or TXT[2] [21/30]
attribute function	e.g.	$L(A16) or $C2(MAT)

The "(scalar-set)" node consists of a set of one or more scalar items, collectively enclosed within parentheses. The delimiter used to separate items may be a comma and/or adjacent blank(s). An item may be a scalar value. For an arithmetic set, an item may also be a range with optional step (default step is 1); e.g. the item 4..10&&2 implies the values 4,6,8 and 10.

```
sname-ssetop-sset >$I IN (51..60)

sname-ssetop-sset >KEY CAS (A XY '1,2')

sname-ssetop-sset >GRADE CSS (A,B)

sname-ssetop-sset >STATUS CNS (ERROR,
WARNING)

sname-ssetop-sset >$I1 NI (7,10,50..200)
```

```
(scalar-set) >(A,B,XY Z '/23')

scalar-set >1 2 5 20 .. 100 && 10

scalar-set >21. .. 35.5

scalar-set >.1 .. .5

scalar-set >.1 ... 5

scalar-set >A,B, I..K

scalar-set >[1.,1.] [0.,6.4] [4.E2,7.]
```

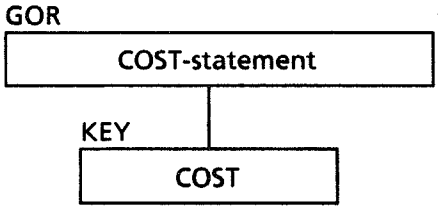


conditions cont'd

COST provides run cost statistics. The kind and format of the statistics are generally dependent upon the host computer and the installation.

ACE > COST

COST



COST-statement

DEFINE ARRAY defines a new IAC array file in the workspace. The example shown defines a 2-D array A.DAT, of size 30 by 50, containing a single double-precision attribute TEMP. Values of TEMP are initialized to zero.

```
ACE > DEFINE ARRAY A.DAT (30,50) (TEMP,R2)
```

The list(arr-index-sizes) node is a list of integers and/or relation specs, one for each array dimension. An integer defines the corresponding dimension size. A relation spec associates that relation with that array index (the relation attributes become labels for the array); the corresponding array dimension size is defined implicitly as the size of the relation.

```
ACE > DEF A A.DAT (30,TIME.DAT) (TEMP,R2)
ACE > DEF A XY.Z (X.REL,Y.REL) (DATA,R1)
```

The referenced relation may be the set of labels associated with an index of an existing array.

```
ACE > DEF A A.DAT (30,B.DAT/2) (TEMP,R2)
```

The core-att-defs node defines attribute(s) in the core array. The simple example shown defines a single matrix, whose attribute CMAT contains 10-character values (values initialized to blank).

```
ACE > DEF A CMAT.DAT (10,10) (CMAT,C10)
```

The core-att-defs node may also be satisified by referencing an existing array or relation file in the workspace. Attribute data from that file are then inserted into the core of the new array. Note that if the referenced file size is too large for the new array, data is truncated; if it is too small, data is extended by insertion of zeros or blanks; in either case, a warning message is generated.

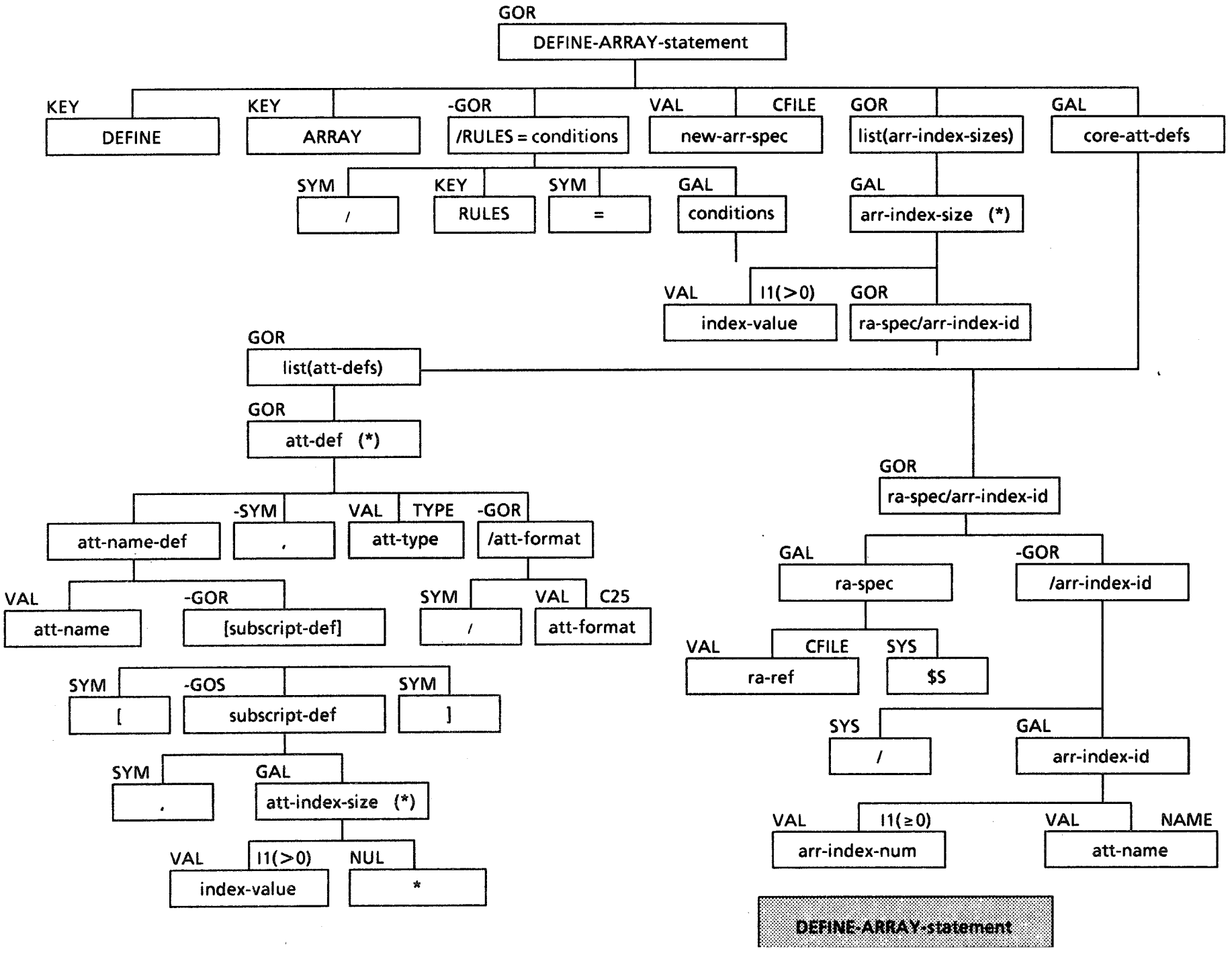
```
ACE > DEF A CMAT.DAT (10,10) OLDMAT.DAT
ACE > DEF A A.DAT (30,B.X) OLDA.DAT
```

For the list(att-defs) node, each attribute definition consists of a name (with optional subscript definition, if non-scalar), an attribute type, and an optional attribute print format.

```
list(att-defs) >(N,I1 X,R1/F10.3)
list(att-defs) >(STRESS[6],R1)
```

Following the DEFINE ARRAY command, you may want to use SHOW FILE to show array characteristics, PRINT to display details of the data, or RELOAD to insert values into the array.

DEFINE ARRAY /RULES = conditions new-arr-spec list(arr-index-sizes) core-att-defs



DEFINE RELATION defines a new IAC relation (table) file in the workspace. The example shown defines a typical mesh relation containing node id and coordinate attributes. Note that a newly defined relation contains no tuples (rows).

```
ACE >DEFINE RELATION NODES.DAT
rel-att-defs >(NODE,I1 X,R1/F10.3 Y,R1/F10.3)

ACE > DEF R NODES.DAT+
(NODE,I1 X,R1/F10.3 Y,R1/F10.3)
```

The rel-att-defs node defines the attributes in the relation. It may contain a list of attribute definitions. The attributes may be of the following types.

```
rel-att-defs >(N,I1 X,R1 C,Z1/2E10.3)
rel-att-defs >(CHAR,C*/'1X,A' LOG,L1)
```

I1 Integer value (1 numeric storage unit)
R1 Real value (1 numeric storage unit)
R2 Real value (2 numeric storage units)
Z2 Complex value (2 numeric storage units)
Z4 Complex value (4 numeric storage units)
L1 Logical value (1 numeric storage unit)
Cn Character value (n character storage units)
C* Character value (variable storage units)

You may also define nonscalar attributes, of any of the above types. Nonscalar attributes may be of arbitrary order and size, and each dimensioned size may be either fixed or variable. Variable size is indicated by an * dimension.

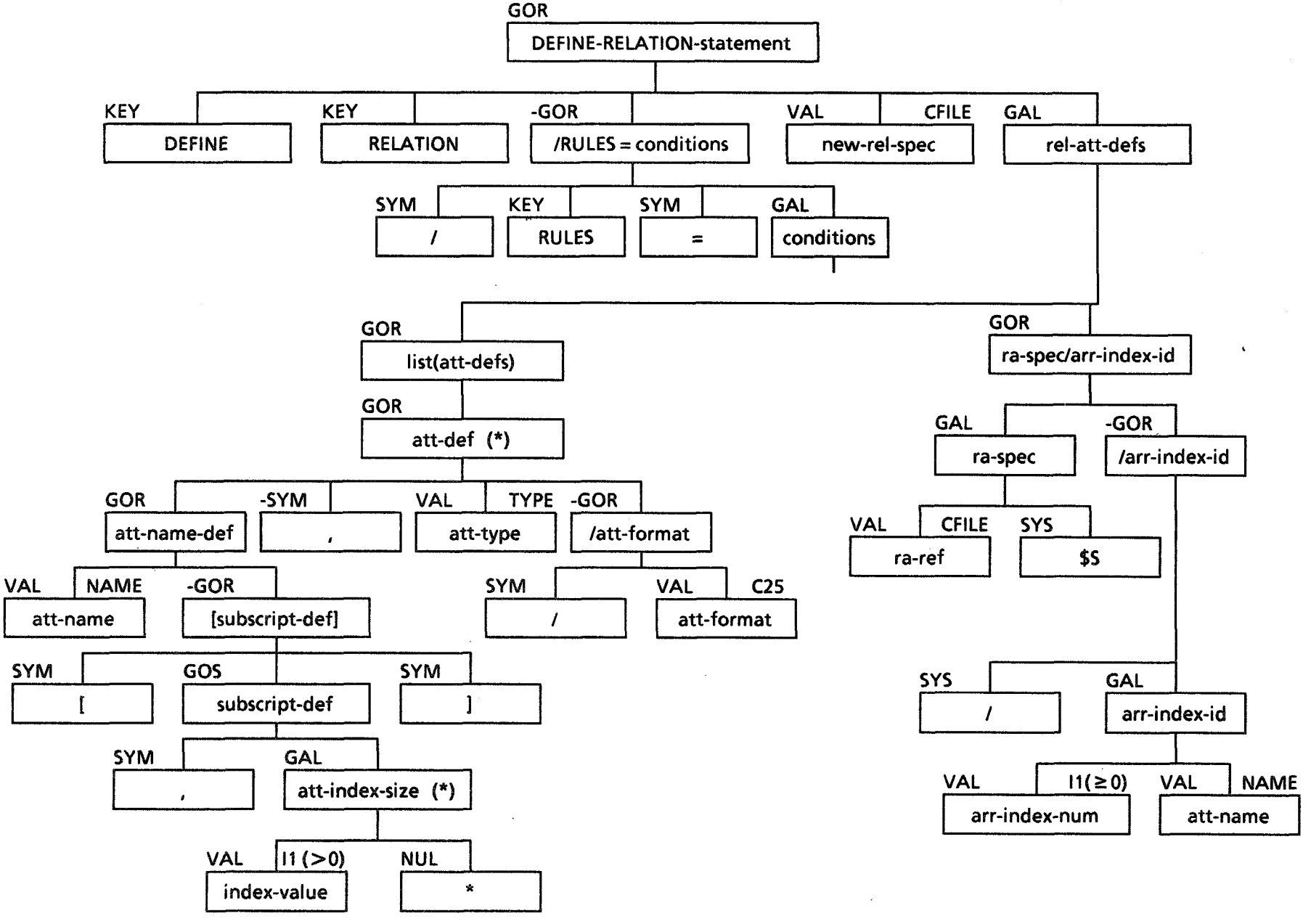
```
rel-att-defs >(STRESS[6],R1/F10.2 LB,I1)

rel-att-defs >(TENSOR4[3,3,3,3],R2/E12.4 +
VAR3[*,3,*],Z2/'2(1X,F10.3)' X[*],R1)
```

Rather than provide a list of attribute definitions, you may save time if you have an existing relation or array in the workspace which already contains the attribute definitions. If you reference this existing file, its attributes will be cloned into the new relation.

```
rel-att-defs> OLD.DAT

rel-att-defs> MATRIX2.DAT/1

rel-att-defs> $S/2
```

Following the DEFINE RELATION command, you may want to use SHOW FILE to show relation characteristics, or LOAD to insert new tuples (rows) of attribute values.

DEFINE RELATION /RULES = conditions new-rel-spec new-rel-def



DEFINE-RELATION-statement

DELETE deletes files from the ACE workspace or from an IAC database. The verify option (default NOVERIFY) lets you confirm the deletion. The version number is a required part of the file spec for the DELETE command.

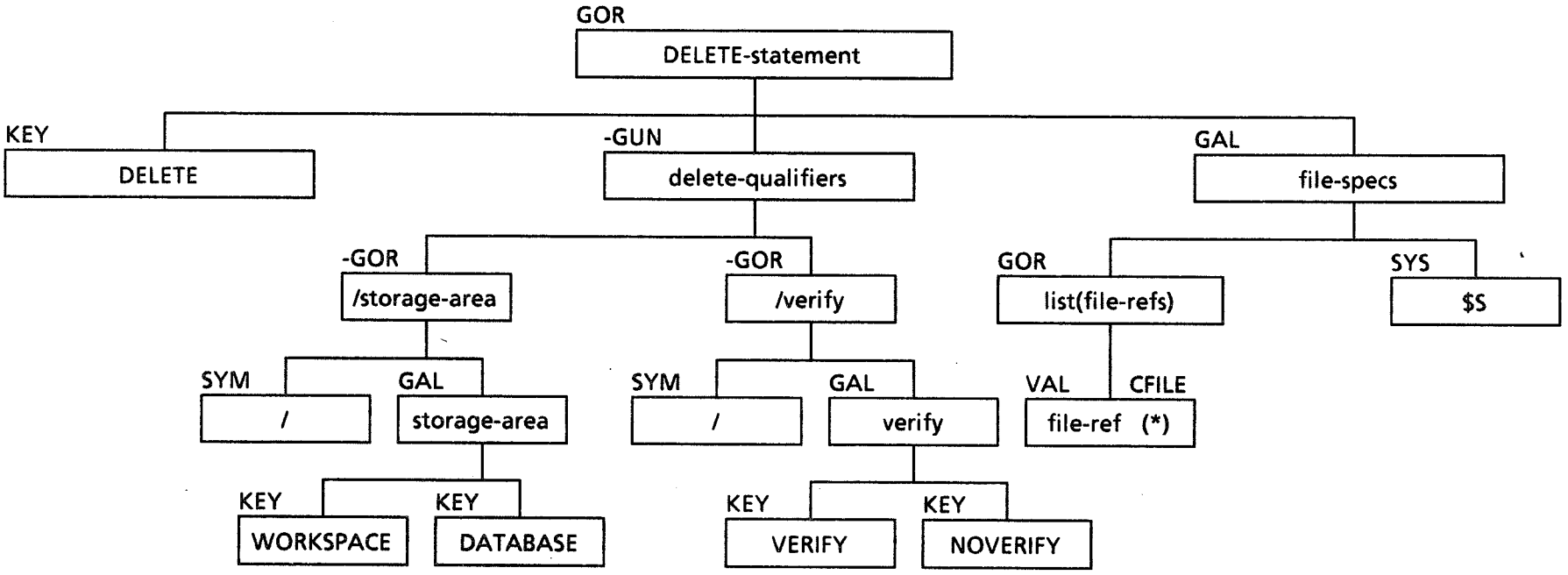
You may use the storage-area qualifier to specify whether files are to be deleted from the WORKSPACE (default) or DATABASE.

As usual, you may use $S to indicate the same list of file-specs as given previously.

Following the DELETE command, you may want to use LIST to list remaining files.

```
ACE >DELETE A.DAT;*
*INFORMATION* Last ACE command processed

ACE >DEL/V A.DAT;1
*INFORMATION* DELETE ... completed
*INFORMATION* Last ACE command processed

ACE >DEL/D X*.*;*

ACE >DEL/D/N *:*.DAT;*


ACE >DEL/W (A.B;1 *.C;* X*.Y;*)

ACE >DEL/D $S
```

DELETE delete-qualifiers file-specs



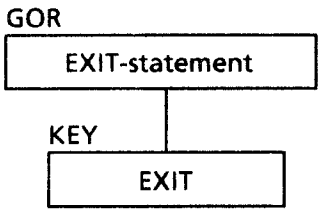
DELETE-statement

EXIT terminates the ACE session by exiting to the host operating system or the next higher level of executing software.

Prior to exiting, be sure to save any files in the workspace that you want to keep, or they will be lost. You may save files with the PUT or PUTS commands.

```
ACE >EXIT
```

EXIT



EXIT-statement

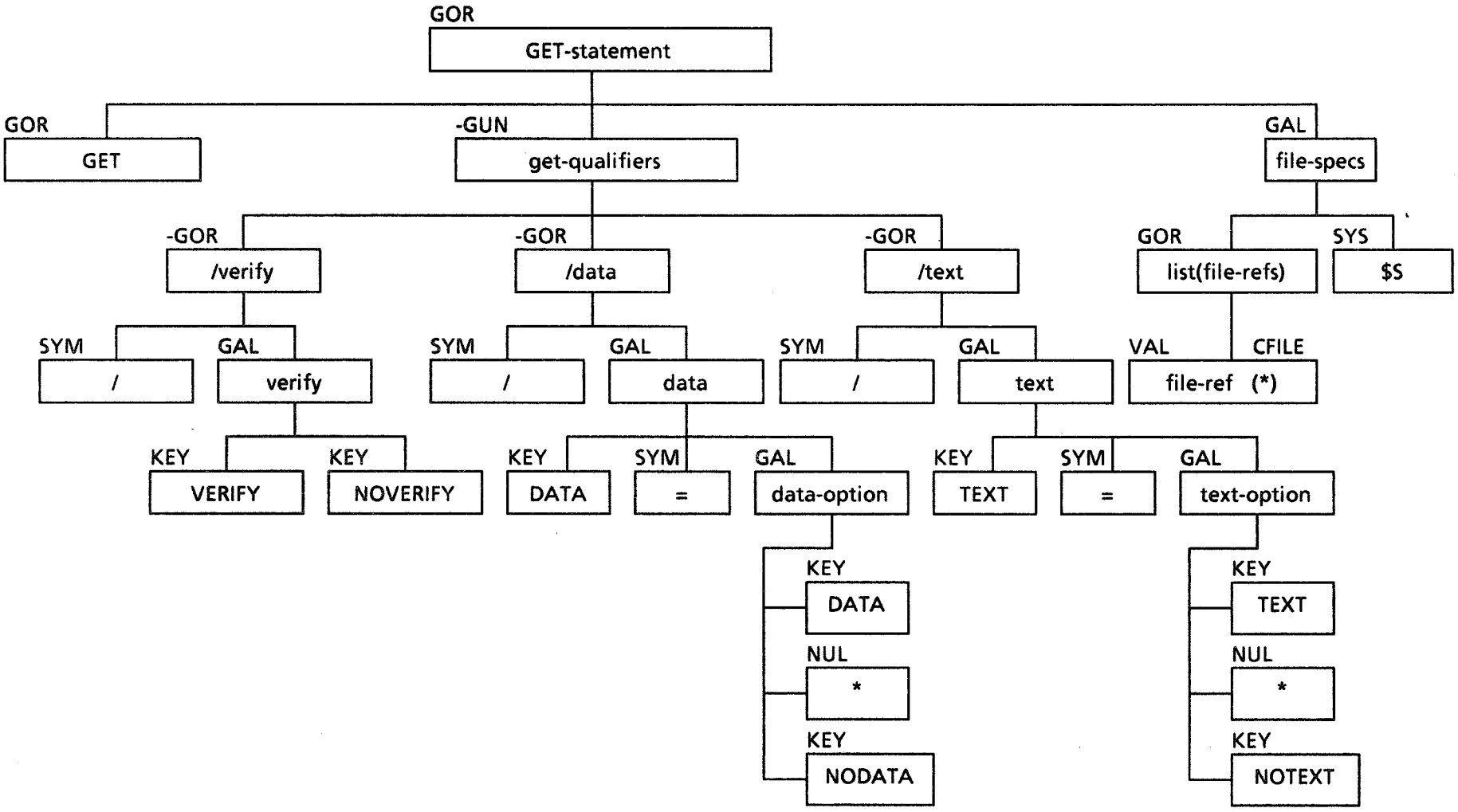
GET allows you to transfer IAC cataloged files from a database to the ACE workspace. The verify option (default NOVERIFY) lets you confirm results of the transfer.

An IAC cataloged file may consist of a data and/or a text file. Note that the host default directory is considered to be a logical extension of the "workspace", and is the area to which a text file or an unstructured data file is transferred. The data-option and text-option qualifiers let you specify whether you want the data and/or text file, respectively, to be transferred. Defaults are DATA=* and TEXT=NOTEXT. The value DATA indicates that the data file is to be transferred, and NODATA does not; * is like DATA, except it indicates that a new specification may be automatically generated for the data file in case there is a collision with an identical existing data file specification. The text-option values are similar to those for data-option.

Following the GET command, you may want to use LIST/W to list files in the workspace, SHOW FILE to show data file structure and organization, or commands such as PRINT to determine detailed data characteristics.

```
ACE >GET A.B,X.DAT
*INFORMATION* Last ACE command processed

ACE >GET/V X.Y
*INFORMATION* GET of database X:1.Y;1
                to workspace X:1.Y;1 completed
*INFORMATION* Last ACE command processed

ACE >GET/T=T PROB3.TEMP

ACE >GET/T=* X.Y

ACE >GET/V/D=D/T=T A.DAT

ACE >GET/D=N/T=* (*.DAT,*.REL)
```

GET get-qualifiers file-specs



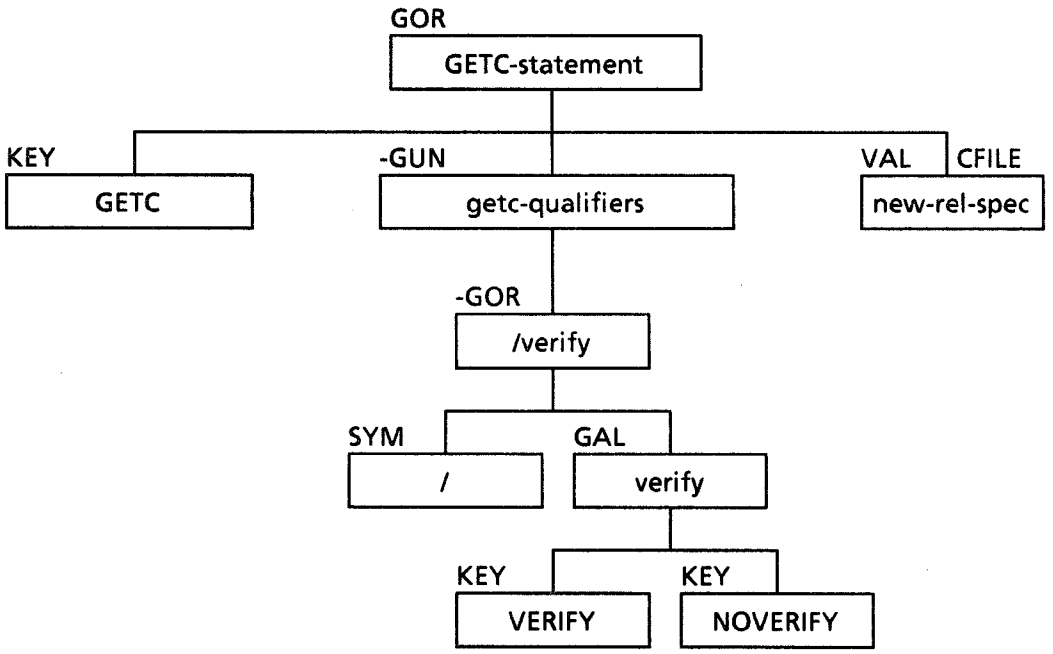
GET-statement

GETC allows you to retrieve a copy of the database catalog and to store it as a relation in the workspace. The verify option (default NOVERIFY) lets you confirm the retrieval. The new-rel-spec is the relation spec by which you want the catalog to be identified in the workspace.

Following the GETC command, you may want to use SHOW FILE to show the catalog relation structure and organization, or commands such as PRINT to determine detailed characteristics.

```
ACE >GETC CAT.TEMP
*INFORMATION* Last ACE command processed

ACE >GETC/V X.Y
*INFORMATION* GETC of database catalog to
                workspace X:1. Y;1 completed
*INFORMATION* Last ACE command processed
```

GETC getc-qualifiers new-rel-spec



GETC-statement

GETG allows you to read a "generic" (ASCII character formatted) data file from a host directory, extract selected data, and create a new relation in the workspace. The verify option (default NOVERIFY) lets you confirm the transfer.

```
ACE >GETG X.DAT X.REL
rel-att-defs >(NODE,I1, X,R2 Y,R2)
*INFORMATION* Last ACE command processed

ACE >GETG/V A.B X.Y (LABEL,C* KEY,I1)
*INFORMATION* GETG of hostfile DB3:[GEN]A.B;1
              to workspace X:1.Y;1 completed
*INFORMATION* Last ACE command processed

ACE >GETS DB2:[PRB.MODEL2]X.DAT;3 Y.REL
rel-att-defs >(NDEX,I1 K,I1 VAL,R1)
```

Note that the GETG command requires two file specs. The first is the spec of the generic file to be read, and may include node and directory specifications. The second is the new IAC relation spec, by which the file is to be identified in the workspace.

The rel-att-defs node defines the new attributes, and its function is identical to that within the DEFINE RELATION command. You may satisfy this node by giving a list of attribute name, type and optional format triplets; alternatively, you may simply give the spec of an existing relation within the workspace, whose attribute definitions are to be cloned for the new relation.

```
rel-att-defs >(STRESS[6],R1/F10.2 LB,I1)

rel-att-defs >OLD.DAT

rel-att-defs >MATRIX2.DAT/1

rel-att-defs >$S/2
```

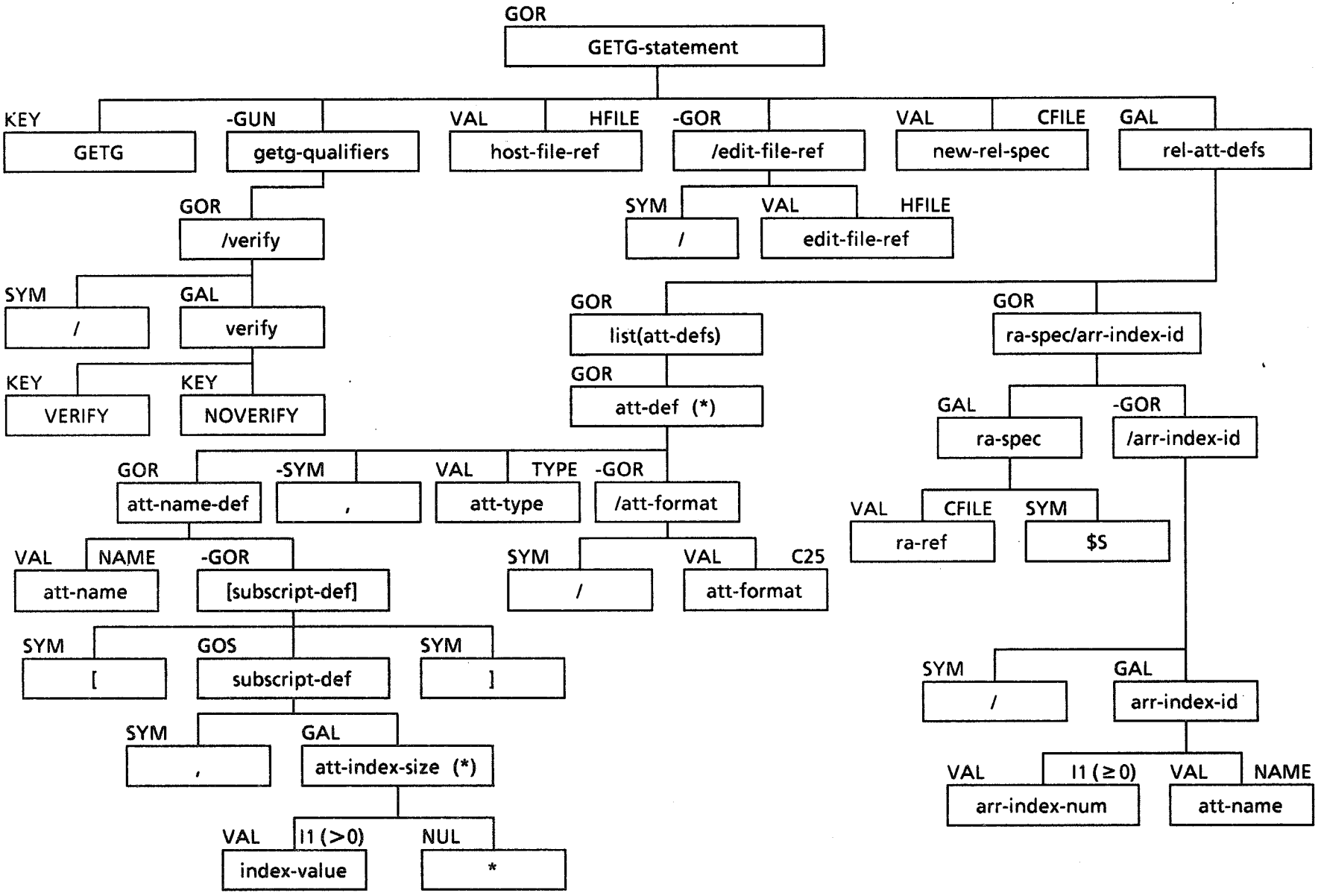
The /edit-file-ref option allows you to supply a card-image file containing one or more nametag options, used to process the generic input file. The option nametags, definitions, and default values are listed at the right. Note that RCONDITIONS and LCONDITIONS are not yet operational. Also see on-line help information for the "/edit-file-ref" node.

```
ACE >GETG A.B /A.EDI X.Y (I,I1 C,R2)
```

nametag	definition	default
RLENGTH	record-length	*
CATENATE	catenate-string	no catenation
CONTINUE	continues-string	no continuation
COLUMNS	column-list	1..*
FIELDS	fields-list	1..*
RCONDITIONS	record-selection	*
LCONDITIONS	line-selection	*

Following the GETG command, you may want to use LIST/W to list files in the workspace, SHOW FILE to show file structure and characteristics, or commands such as PRINT to determine detailed characteristics.

GETG getg-qualifiers host-file-spec /edit-file-spec new-rel-spec new-rel-def



GETG-statement

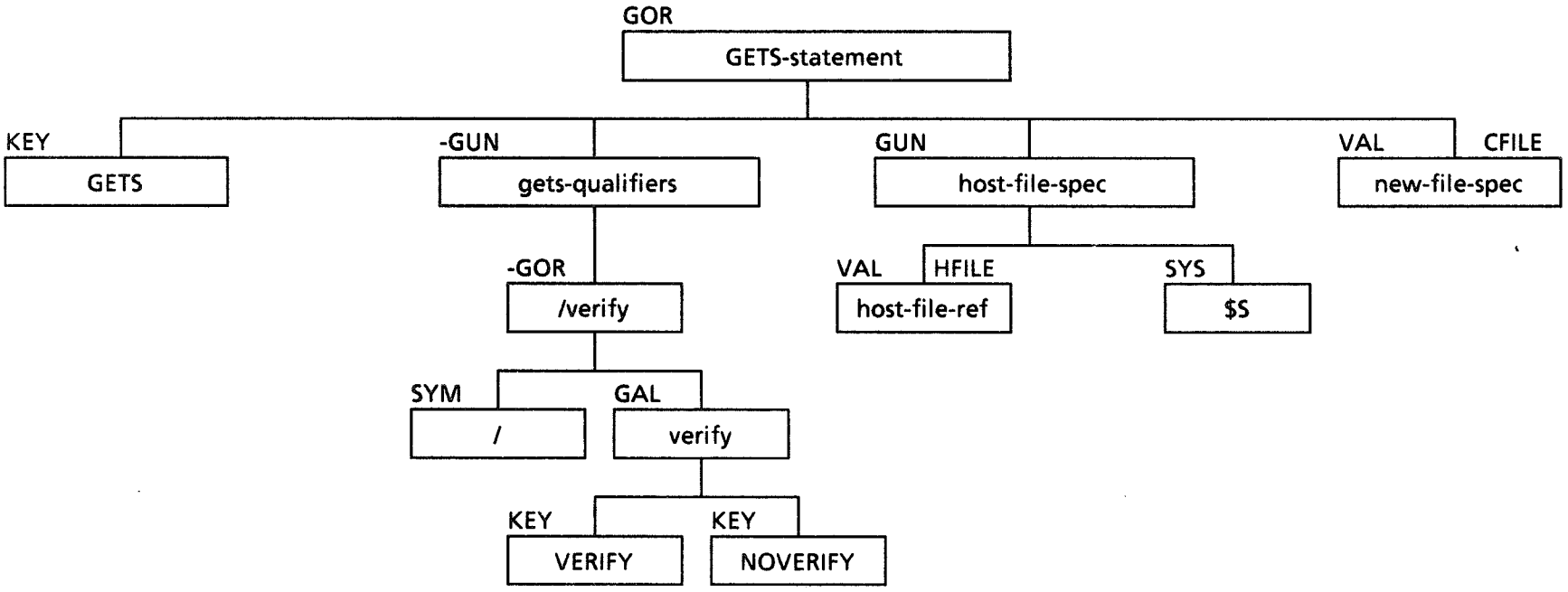
GETS allows you to transfer an IAC structured data file (i.e. a relation or array) from a host directory to the ACE workspace. The verify option (default NOVERIFY) lets you confirm the transfer.

Note that the GETS command requires two file specs. The first is spec of the host file to be retrieved, and may include node and directory specifications. The second is the new IAC cataloged file spec, by which the file is to be identified in the workspace.

Following the GETS command, you may want to use LIST/W to list files in the workspace, SHOW FILE to show data file structure and organization, or commands such as PRINT to determine detailed characteristics.

```
ACE >GETS A.DAT A.DAT
*INFORMATION* Last ACE command processed

ACE >GETS/V X.Y  X.Y
*INFORMATION* GETS of hostfile DB1:[TLC]X.Y;1
                 to workspace X:1.Y;1 completed
*INFORMATION* Last ACE command processed

ACE >GETS DB2:[MAIN.MODEL2]X.DAT;3 Y.DAT

ACE >GETS A.DAT X:1.DAT
```

GETS gets-qualifiers host-file-spec new-file-spec



GETS-statement

HISTOGRAM generates a printer bar chart on the currently defined OUTPUT file. The chart describes the distribution of values for the selected tuples of a specified attribute in a relation or array. Basic summary statistics are also displayed. Histograms can be generated for integer, real and double precision scalar values.

```
ACE >HISTOGRAM A.DAT X *
ACE >HIST A.DAT STRESS[3] *
ACE >HIST C.D. M3 M3 IN (50..120)
```

The conditions define the subset of values to be selected. The example shown provides a histogram for the diagonal elements of a matrix.

```
ACE >HIST MATRIX.DAT A $I1 EQN $I2
```

Optional histogram qualifiers include sort order, barwidth and symbol (respective defaults are LOHI, value range divided by square root of number of selected tuples, and *). These qualifiers are not often used, but may be convenient for special cases. The sort order controls sorting of values for output; the barwidth defines the value range represented by each bar; and the symbol defines the print character to be used in forming the bars.

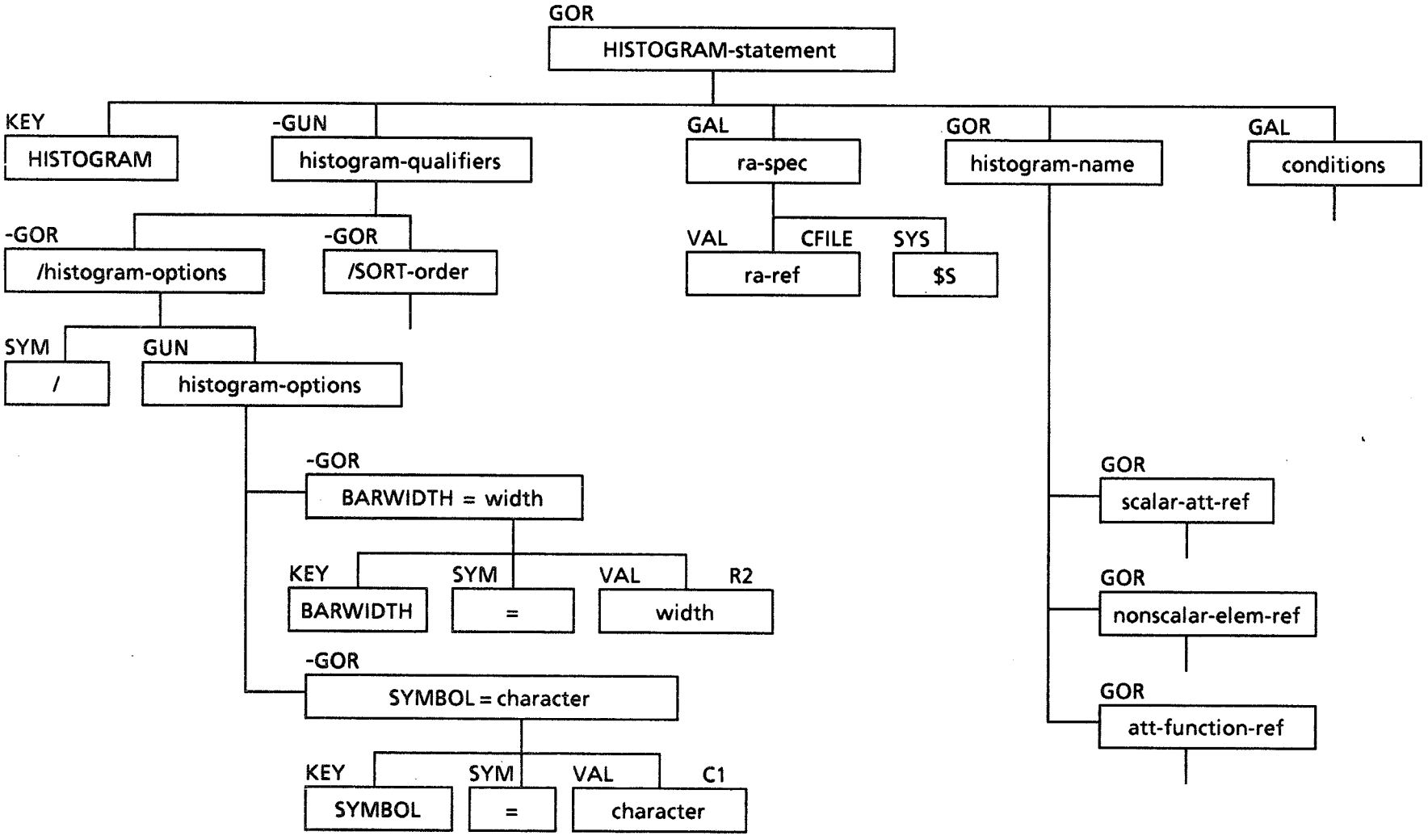
```
histogram-qualifiers >/HILO
histogram-qualifiers >/BARWIDTH=10
histogram-qualifiers >/HILO/SYM=X
```

This histogram-name indicates the scalar value to be displayed. This name may designate a scalar attribute, an element of a nonscalar, or an attribute function.

```
histogram-name >X
histogram-name >STIFF[3,3]
histogram-name >$REAL(CMPLXVAL)
```

Prior to using the HISTOGRAM command, you may want to use SHOW FILE to show data file structure and organization, or commands such as TALLY or PRINT to determine basic characteristics.

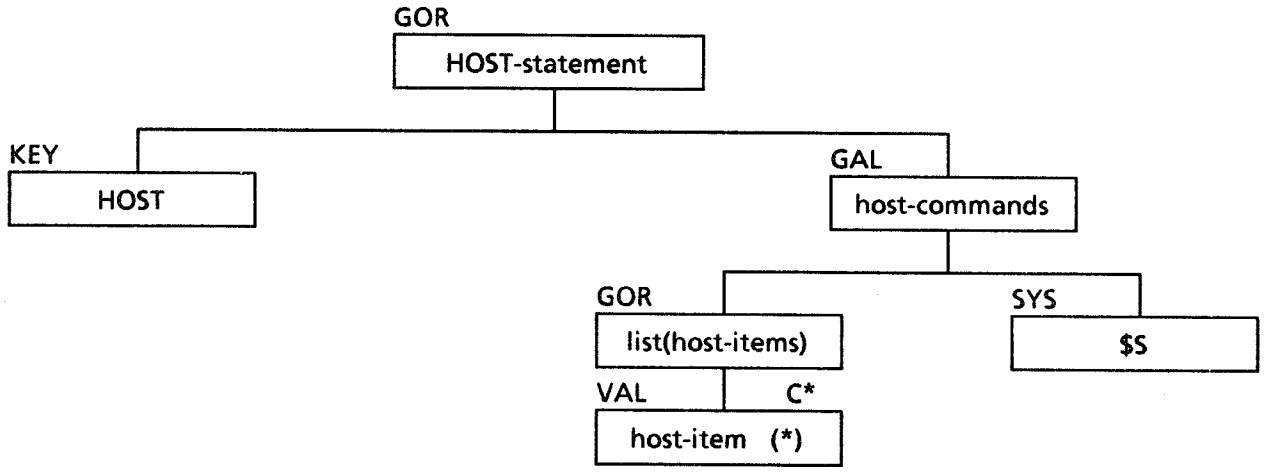
HISTOGRAM HISTOGRAM-qualifiers ra-spec name



HOST provides a mechanism by which you can execute any host operating system command(s) from within ACE. You specify each host command as a string enclosed within apostrophes. Some host commands, e.g. those involving a host editor, may initiate interaction with you. In any case, when the host command operations are completed, control is returned to you within ACE.

```
ACE >HOST '$DIR'

ACE >'$COPY A.B. X.Y', '$DIR'

ACE >HOST '$EDIT'
```

HOST host-commands



HOST-statement

JOIN horizontally joins two relations to form a new relation.

```
ACE >JOIN A.REL B.REL NODE AB.REL
```

The set of attributes (columns) in the new relation consists of all attributes in the first relation, plus each attribute in the second relation whose name does not duplicate a name in the first relation.

The set of tuples (rows) in the relation is determined by matching values corresponding to the "join-names" node. A new tuple is created wherever a match occurs. In the first example on the right, the NODE attribute in the first relation is compared with the NODE attribute in the second relation, and a new tuple is created wherever the values are equal. If the corresponding attributes to be matched have different names, you give both names separated by a "/". This is shown by the second example; note that in this case both NODE and GRID will appear as attributes in the new relation. You may give multiple join names, as shown in the third example; then a new tuple is created only if all the join names match.

```
join-names >NODE

join-names >NODE/GRID

join-names >KFLAG, NODE/GRID, LABEL
```

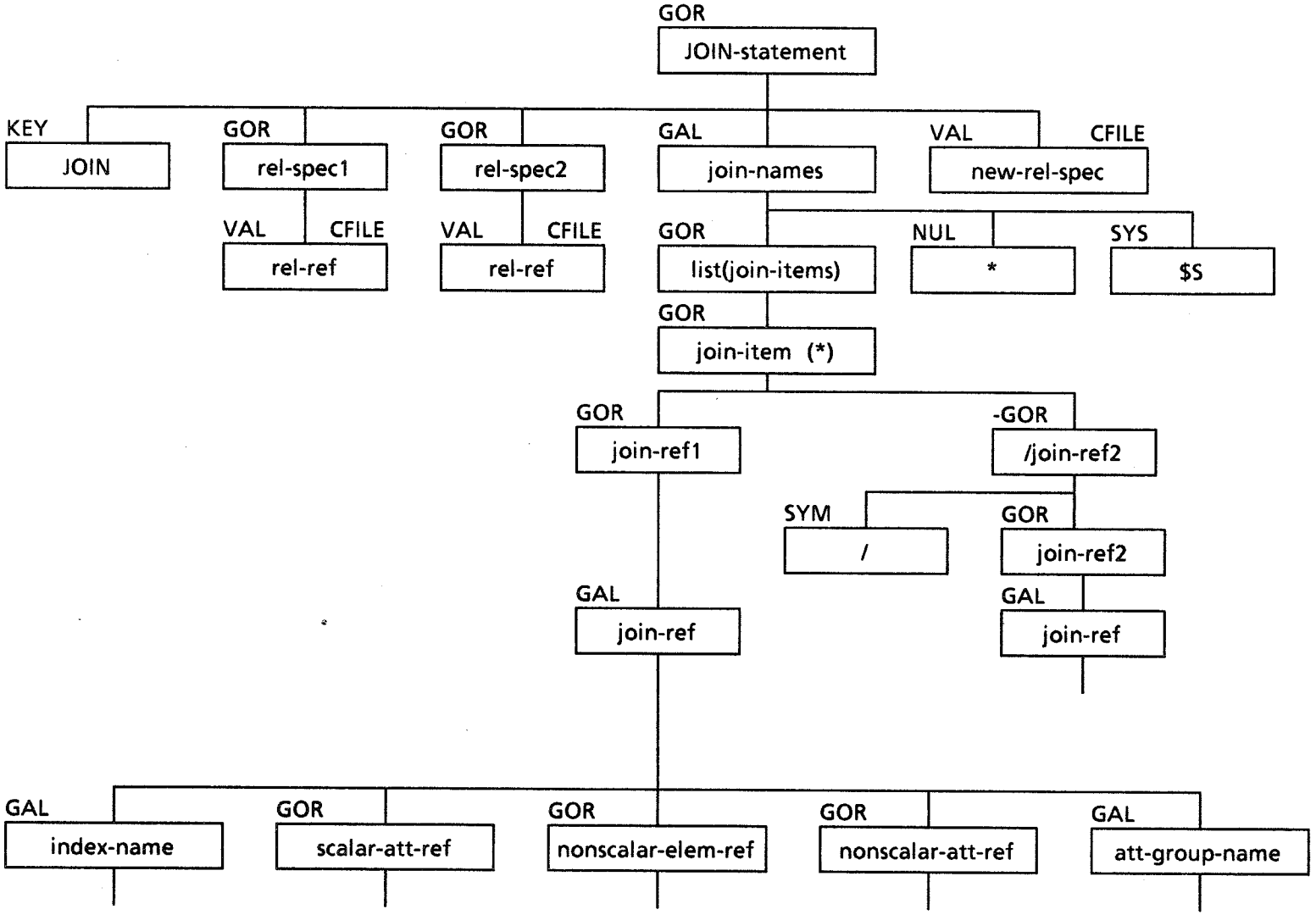
You may use various kinds of join-names. For example, the index-name $I creates a new relation whose number of tuples is the smaller of those in the two existing relations. If you include a nonscalar in the list of join names, all respective specified elements are compared.

```
join-names >$I

join-names >NDEX[2], LABEL

join-names >NDEX[2]/$I

join-names >NDEX[3..4]/NDEX [1..2]
```

Finally, you may use the null token "*" to satisfy the join-names node. You should use caution here, since the number of tuples in the new relation is the product of the numbers of tuples in the two existing relations! As an example, you might use this to join a constant value, given in a single-tuple relation, onto every tuple of another relation.

```
join-names >*
```

JOIN rel-spec1 rel-spec2 join-names new-rel-spec



JOIN-statement

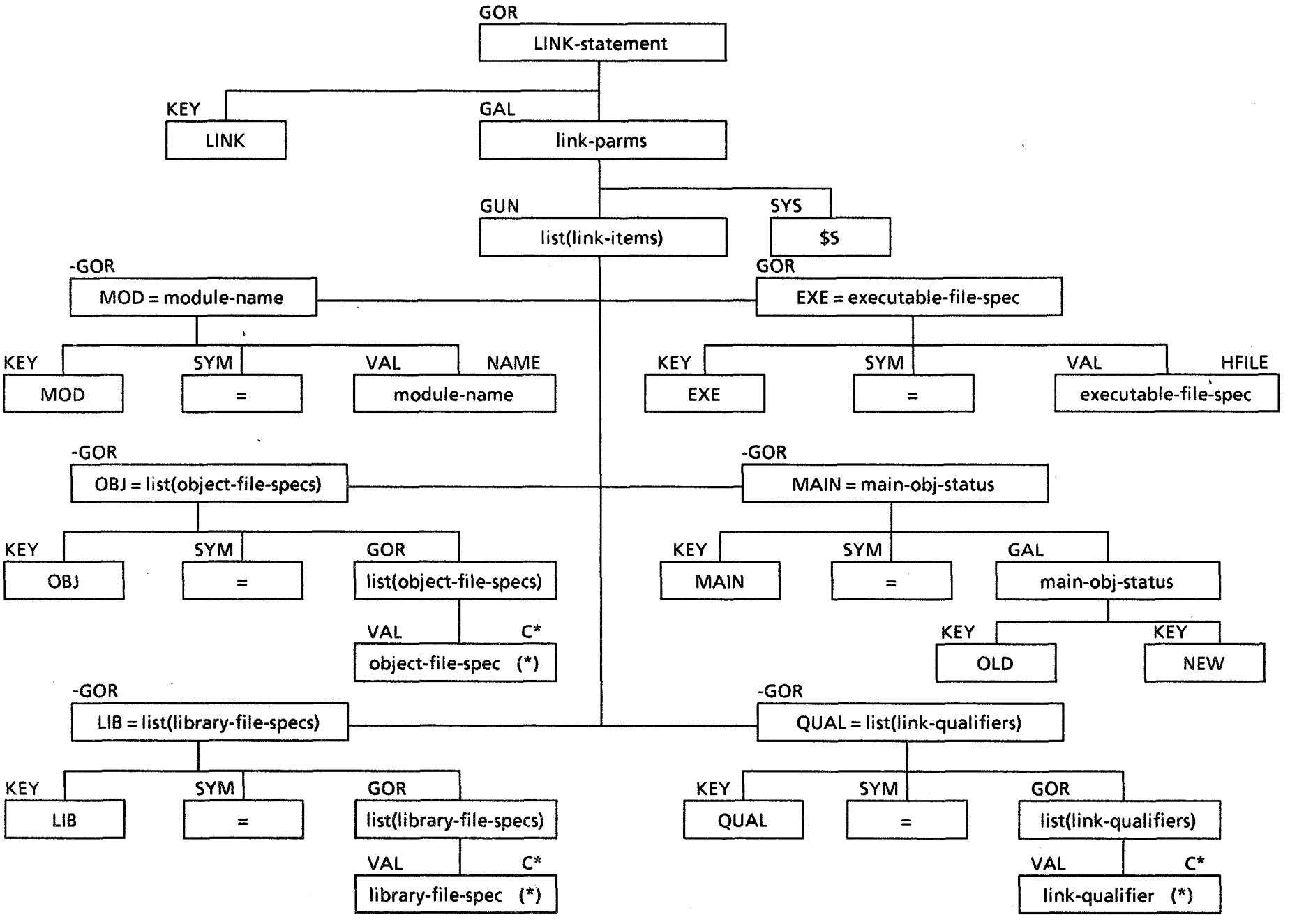
LINK allows you to link (i.e. to form an executable) for your own version of a module. If you LINK one of the IAC modules, you may modify and/or add your own subroutines; ACE will automatically link in the standard module routines which you have not replaced, and any required standard IAC libraries. If you LINK one of your own modules, use the MOD = USER parameter. In this case you must supply the main program and all required subroutines in the form of object files or libraries; however, ACE will automatically link in any required standard IAC libraries.

The MAIN parameter specifies whether you are using the standard OLD main program (default), or you are providing your own NEW version of the main program.

The QUAL parameter allows you to include a list of special host dependent link qualifiers. These are simply passed on as character strings to the host linker.


```
ACE >LINK (MOD=DISCOS, EXE=DIS3,+
OBJ= CONTROL, KHINGE)

ACE >LINK (MOD=USER, EXEC=MYMOD,+
MAIN=NEW, OBJ=MYPROG,MYSUB,ABC,+
LIB='MYLIB/LIB')

ACE >LINK (MOD=USER,MAIN=N,EXE=Z,+
OBJ=X, '[OTHERDIR]Y')
```

LINK link-parms



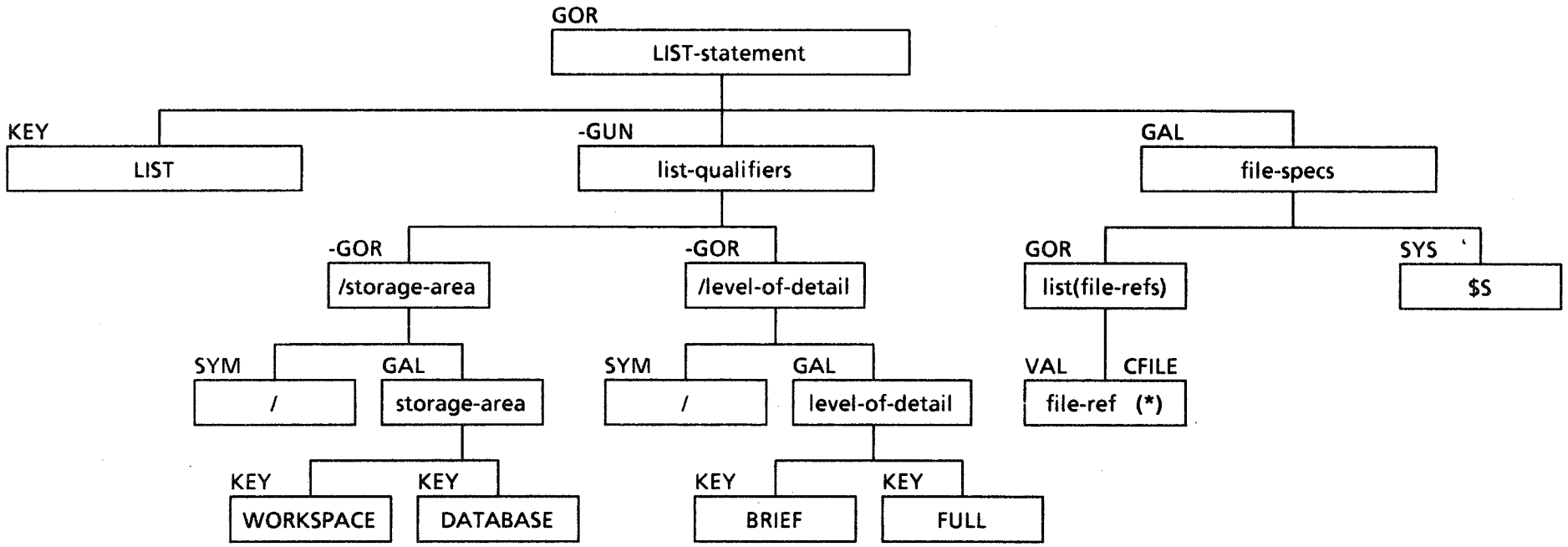
LIST gives you catalog information for a set of IAC cataloged files. These files may exist either in the WORKSPACE (default) or DATABASE storage.

You may request either a BRIEF (default) or FULL information level of detail. FULL displays all attributes in the catalog.

Note that $S, as usual, is shorthand for the last value input for the corresponding node.

```
ACE >LIST *.*

ACE >LIST A.DAT,*.X,NODES:1.*

ACE >LIST/D *.DAT

ACE >LIST/F *.*

ACE >LIST/F/D (*.DAT *.X)

ACE >LIST/B (A.DAT, *.X,NODES:1.*)

ACE >LIST/F $S
```

LIST LIST-qualifiers file-specs



LOAD allows you to load tuples into an existing relation file. After the LOAD command syntax is processed, ACE will give you interactive assistance while you input values. The optional "/start-index-value" node specifies the starting tuple number for loading (default is 1 + maximum existing tuple number). The $ symbol terminates loading.

```
ACE >LOAD A.DAT

ACE >LOAD A.DAT /131
```

You insert the new values via data input lines following the LOAD command line. Whenever another data value is needed, ACE provides you with a prompt describing the current tuple number and attribute name.

```
ACE >LOAD NODES.DAT
Tuple=1 NODE >

Tuple=1 NODE >13
Tuple=1 X > 1.3  10.0
Tuple=1 Z >
```

If nonscalar attributes are loaded, the ACE prompt also includes the current index position within the attribute. Input must include a parentheses level for each index level (ACE provides the parentheses for you, whenever it can).

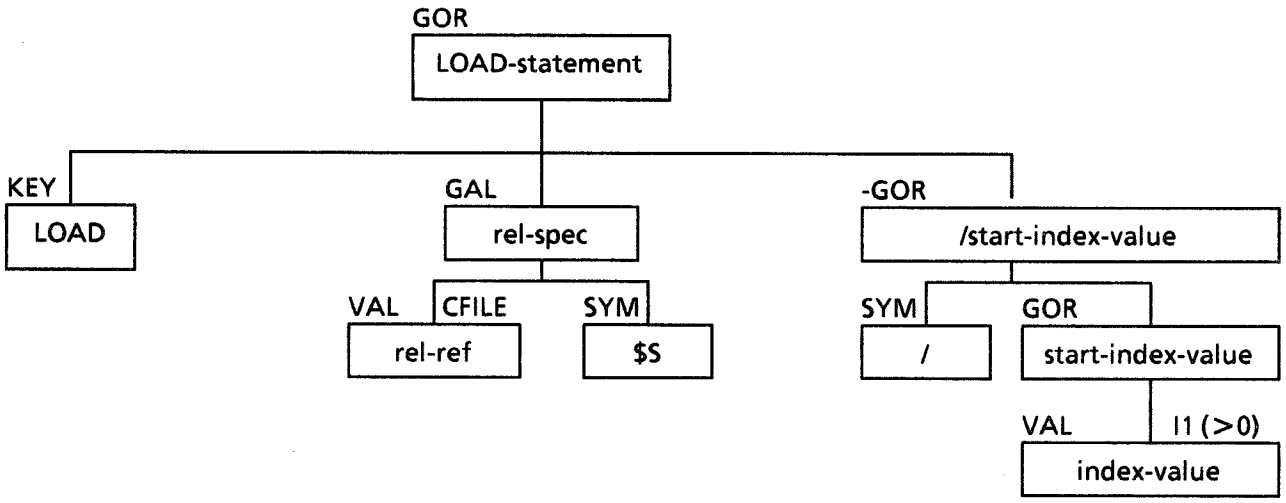
```
Tuple=73 MATP[1,1] > ((1.0 1.7, 1.5)
Tuple=73 MATP[1,2] > (6.3, 2.8)
Tuple=73 MATP[1,3] >
```

A data line with at most blank(s) causes the current attribute position to be repeated, while a data line ending in a dollar sign (i.e. $) causes loading to terminate. If your input line begins with a percent (i.e. %), then the line is a control line. Control lines of %H and %E respectively display help information and examples.

```
Tuple=3 X > 1.5 6.3
Tuple=3 Z > 5.1 $

Tuple=84 DAT3 > %H

Tuple=84 DAT3 > %HE
```

Following the LOAD command, you may want to use SHOW FILE to show data file structure and organization, commands such as PRINT to determine detailed characteristics, or RELOAD or CHANGE to modify existing values.

LOAD rel-spec /start-index-value



LOAD-statement

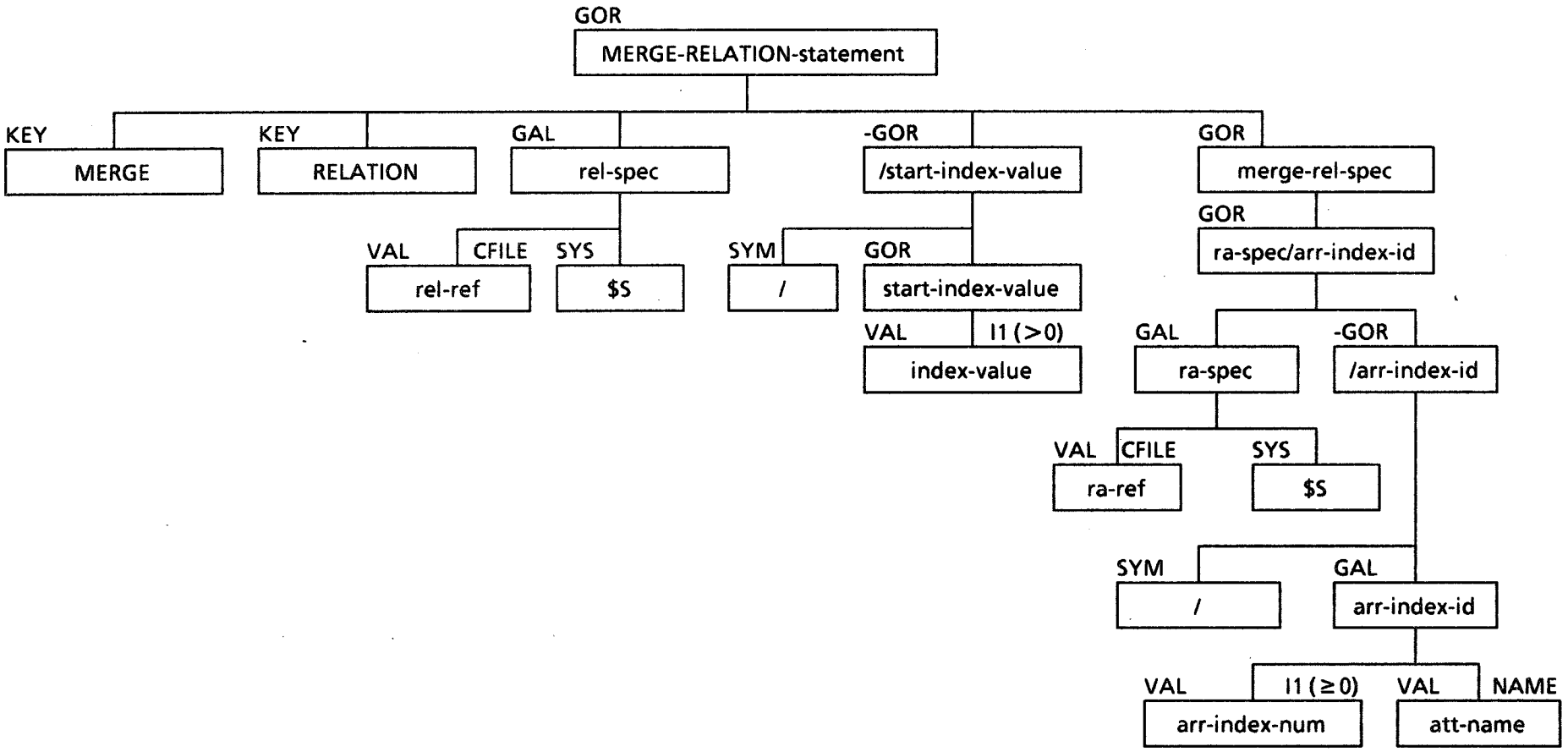
MERGE RELATION vertically expands a relation by merging in another (merge) relation. All corresponding attributes from the two respective relations must agree in name, dimension and type. The "/start-index-value" node allows you an option to insert the merge relation starting at a specified tuple number; default is the end of of the relation, i.e. at index value = 1 + maximum tuple number.

```
ACE >MERGE RELATION A.DAT B.DAT

ACE >MERGE RELATION A.DAT/51 B.DAT
```

The "merge-rel-spec" node offers you some flexibility as to where the merge relation is obtained. A simple relation spec indicates the specified relation. An array spec followed by a qualifying index id indicates that the merge relation consists of the set of attributes associated with that index of the array. The index id is either an index number, or an attribute name which implies the associated index number.

```
merge-rel-spec >NODES.REL

merge-rel-spec >TEMP.ARR/1

merge-rel-spec >TEMP.ARR/NODE

merge-rel-spec >MAT.DAT/0
```

MERGE RELATION rel-spec /start-index-value merge-rel-spec



OPEN is used to open an old or new IAC database. A database must be opened before it can be accessed by commands such as GET, PUT, GETC, LIST/D, DELETE/D and PURGE/D.

Note that a database corresponds to a host directory (or sub-directory). The database contains an activities file, a catalog file, and those files which are listed in the catalog. Only one database may exist in the directory, but the directory may contain non-database files.

OPEN has four parameters: directory and username are required, and status (default OLD) and access (default CONCURRENT) are optional. Typical OPEN commands for an OLD database are as shown.

```
ACE >OPEN (U=JONES, D='[FBJ.DB]')

ACE >OPEN
list(OPEN-parms) >(D='[MAIN]',
USERNAME=username >USERNAME=BOB)
```

The status indicates whether the database is OLD (existing) or NEW (to be created). Note that in either case, the directory must exist before the database is opened. That is, NEW simply creates the database activities file and catalog file on the specified existing directory. You can easily create a sub-directory for a new database by using the HOST command within ACE. The new username becomes the database owner.

```
ACE >HOST '$CREATE/DIR [MAIN.X.DBASE]'
*INFORMATION* Last ACE command processed
ACE > OPEN(U=AMV,D='[MAIN.X.DBASE]',
STATUS=status >STATUS=NEW)
```

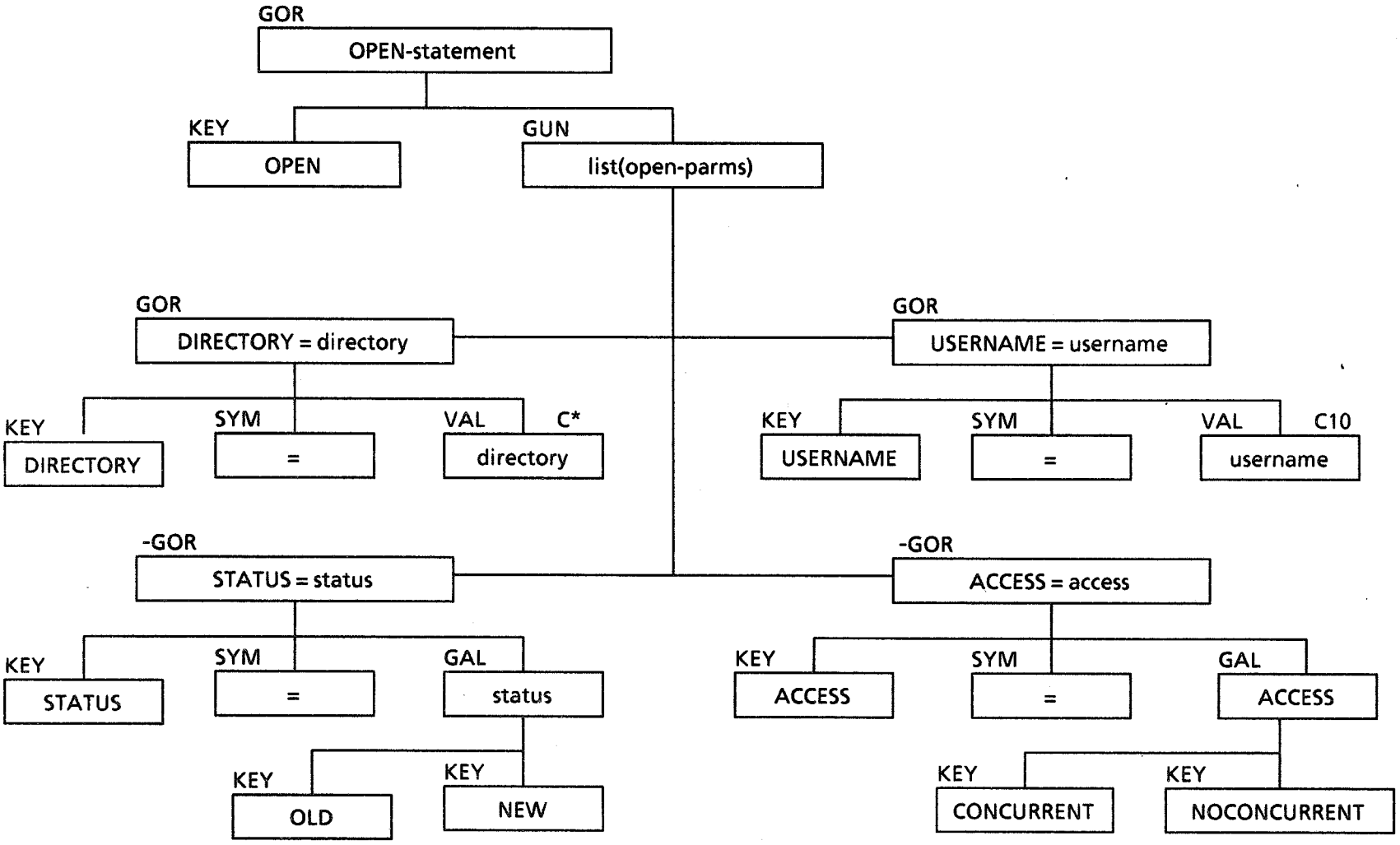
You will generally want to take the default CONCURRENT value for access. This protects you against possible undesirable side effects, due to other users or concurrent interactive/batch tasks. However, NOCONCURRENT does provide somewhat more efficient access to the database.

```
ACE >OPEN(D='[MAIN]',U=TOM,A=NO)
```

Note that an OPEN command automatically closes any database open as a result of a previous OPEN command. A series of OPEN commmands may be used to transfer data between databases.

```
ACE >BEGIN OPEN (D='[X.Y1]',U=BOB) +
GET*.* OPEN(D='[X.Y2]',U=BOB) +
PUT *.* END
```

OPEN list (open-parms)



OPEN-statement

PLOT allows you to create an ASCII formatted plot file from data stored within a relation. You can then use the PLOT module to create various kinds of X-Y Cartesian or polar plots, bar charts, etc. Since the PLOT command provides many detailed options, you will probably want to start with only the required entities, as shown in the first example. As shown, only the relation spec, selected attribute names, and host plot file spec are given. Note that any of the other optional entities may be supplied by you later, as you interactively execute the PLOT module.

```
ACE >PLOT BEGIN
plot-relation >RELATION GOOD.DATA
plot-names >NAMES (XDAT,YDAT,Z)
plot-hostfile >HOSTFILE GOOD.DAT END
```

However, if you want to be prompted for all the options within ACE, you may proceed in a manner similar to that shown in the example to the right. Note that various keywords (RELATION, NAMES, HOSTFILE, etc.) are provided for you by ACE as auto inputs. To reject any option, you as usual just hit the carriage return.

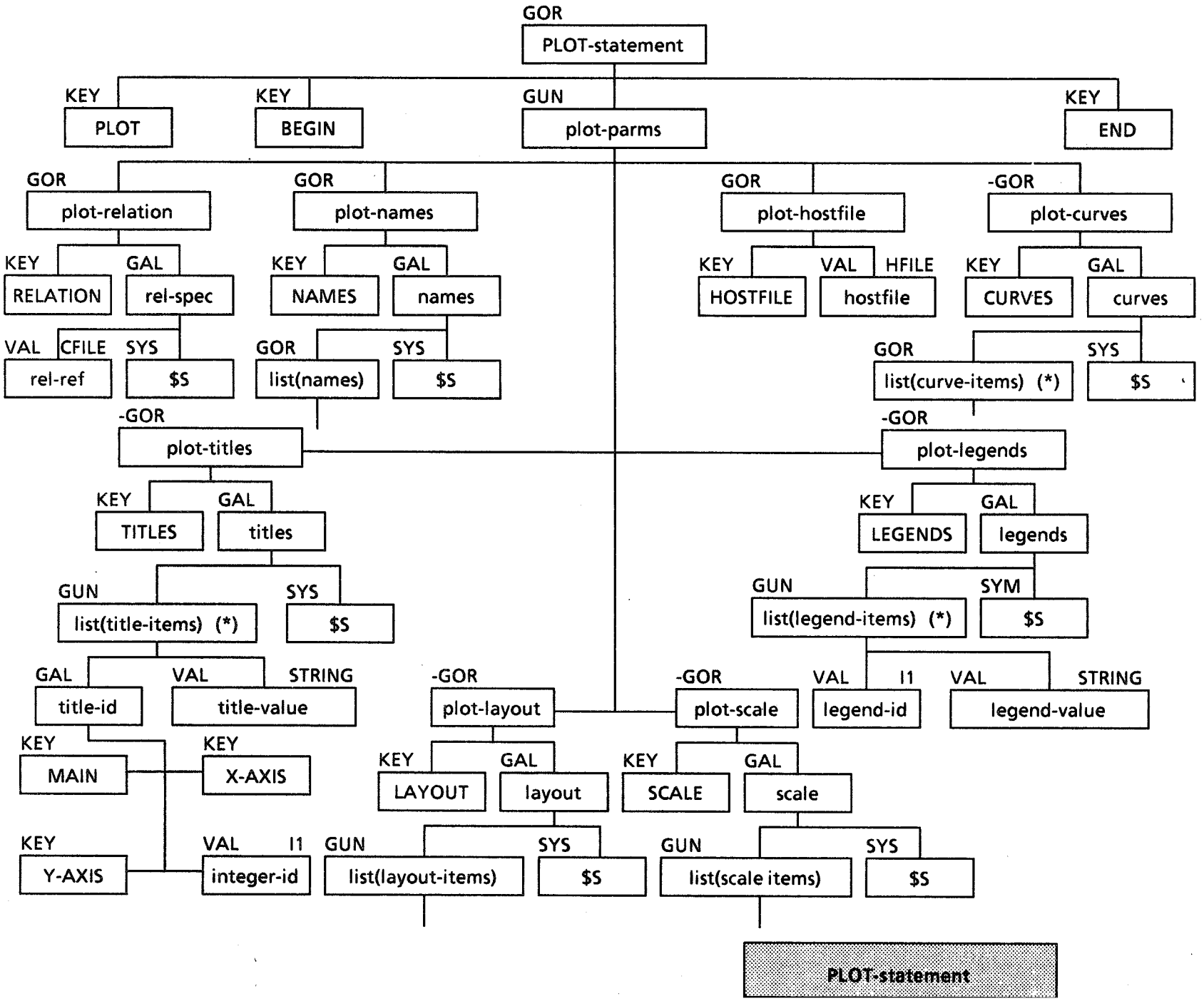
```
ACE >PLOT BEGIN
plot-relation >RELATION A.DAT
plot-names >NAMES (TIME V1 V2 T3)
plot-hostfile >HOSTFILE A.DAT
plot-curves >CURVES
plot-titles >TITLES
plot-legends >LEGENDS
plot-layout >LAYOUT
plot-scale >SCALE
END >END
```

You may also find a third alternative useful, in which you supply a detailed PLOT command within an ACE command file (see INPUT option in the SET IO command). The example at the right is a listing of such a file. It is assumed for purpose of illustration that that all options can be predefined in terms of fixed values, except for the main title which is to be provided via an amacro prompt (see Section 2.2). Other options could of course be made variable in a similar manner using amacro prompts. Note that the legends are text strings which may be interactively selected and positioned on the plot.

```
PLOT BEGIN +
RELATION STAGE.VEL +
NAMES TIME, VEL1, VEL2 +
HOSTFILE TEST DATA.PLT +
CURVES    (1 X-AXIS=TIME Y-AXIS=VEL1) +
          (2 X-AXIS=TIME Y-AXIS=VEL2) +
TITLES    (MAIN ? main title ?) +
          (X-AXIS 'Time in seconds') +
          (Y-AXIS 'Stage velocities') +
LEGENDS   (1 '2nd stage') +
          (2 '3rd stage') +
LAYOUT (X-AXIS=LIN, Y-AXIS=LOG) +
SCALE X DIVISIONS =0.5 +
END
```

Note that integer, real, complex, logical and character values may be plotted. See Appendix B for a detailed description of all plot file options.

PLOT BEGIN plot-parms END



Each "list(curve-items)" node defines a curve; the definition consists of a curve id (you may later reference this within the PLOT module) and associated parameter(s).

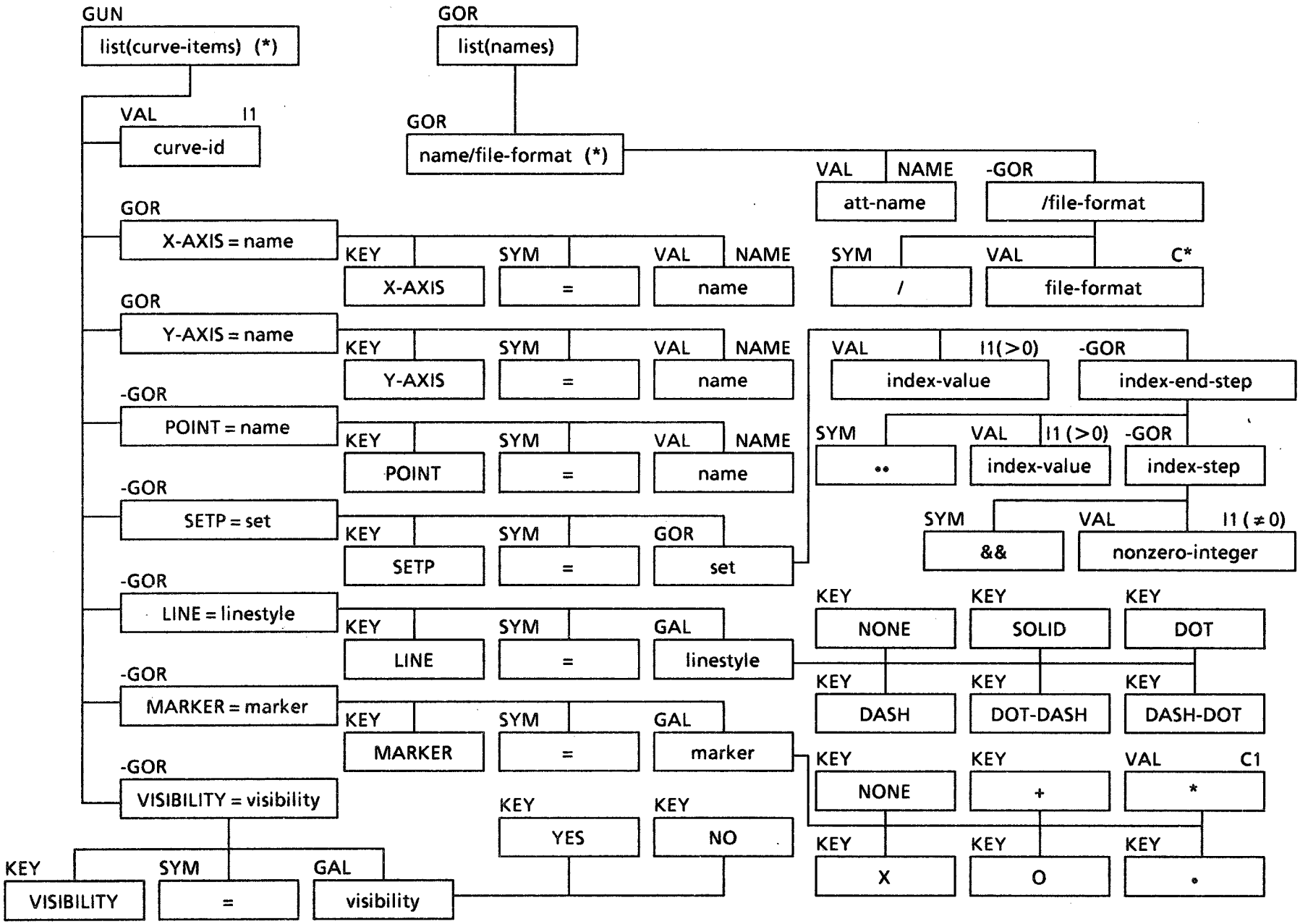
The X-AXIS and Y-AXIS parameters define the attribute names to be associated with the respective axes. Note that for polar coordinates, X and Y mean radial and circumferential. POINT allows you to specify an additional attribute name, whose values are to be associated with and displayed at the curve point positions themselves. Note that X-AXIS, Y-AXIS and POINT collectively allow the display of 3-variable relationships. SETP allows you to specify a subset of POINT values to be displayed.

LINE and MARKER define the respective line style (default SOLID) and marker symbol (default NONE) to be used for the curve.

Visibility of the curve may be specified as either YES (on, default) or NO (off). You may want to have a number of curves defined initially with VISIBILITY=NO, and then make selected curves visible for particular display combinations within the PLOT module.

The "list(names)" node gives a list of attribute names and optional associated formats. The given attributes are those whose values are to be placed on the plot file. The associated format allows you to specify a FORTRAN write format, for writing the values on the plot file. Note that the format for writing values on a plot (associated with the POINT option) comes from the usual attribute print format.

```
list(curve-items) >(3,X=TIME,Y=VELOCITY)
curve-id >3

X-AXIS=name >X-AXIS=TIME
Y-AXIS=name >Y-AXIS=DECAY_RATE
POINT=name >POINT=I
SETP=set >SETP=10..200&&10

LINE=LINESTYLE >LINE=DOT-DASH
MARKER=marker >MARKER=X
MARKER=marker >MARKER='*'
VISIBILITY=visibility >VISIBILITY=NO

list(names) >(A,BX,CMAX)
list(names) >(A,BX/'F6.1',CMAX)
```


GUN
list(curve-items) (*)
VAL I1
curve-id
GOR
X-AXIS = name
KEY SYM VAL NAME
X-AXIS = name
GOR
Y-AXIS = name
KEY SYM VAL NAME
Y-AXIS = name
-GOR
POINT = name
KEY SYM VAL NAME
POINT = name
-GOR
SETP = set
KEY SYM GOR
SETP = set
-GOR
LINE = linestyle
KEY SYM GAL
LINE = linestyle
-GOR
MARKER = marker
KEY SYM GAL
MARKER = marker
-GOR
VISIBILITY = visibility
KEY SYM GAL
VISIBILITY = visibility
KEY KEY
YES NO
GOR
list(names)
GOR
name/file-format (*)
VAL NAME -GOR
att-name /file-format
SYM VAL C*
/ file-format
VAL I1(>0) -GOR
index-value index-end-step
SYM VAL I1 (>0) -GOR
•• index-value index-step
SYM VAL I1 (≠ 0)
&& nonzero-integer
KEY KEY KEY
NONE SOLID DOT
KEY KEY KEY
DASH DOT-DASH DASH-DOT
KEY KEY VAL C1
NONE + *
KEY KEY KEY
X O •

The "list (layout-items)" node defines the plot layout, via several optional parameters.

```
list (layout-items) >(X=LOG, Y=LIN, C=POLAR)
```

The X-AXIS and Y-AXIS parameters define the respective axis types. Choices are CONSTANT (e.g. for bar chart), LINEAR (default) and LOGARITHMIC.

```
X-AXIS=axis-type >X-AXIS=LOG
```

BACKGROUND defines the resolution of the background grid. Choices are STANDARD (default), FINE and NONE.

```
BACKGROUND=background-type >BACKGROUND=FINE
```

COORDINATES defines the plot coordinates. Choices are CARTESIAN (default) and POLAR. Note that for polar coordinates, X and Y mean radial and circumferential (counter-clockwise degrees).

```
COORDINATES=coordinates-type >COORDINATES=P
```

The "list(scale-items)" node defines the plot scale characteristics, via several optional parameters.

```
list (scale-items) >(XMAX=5.5, YDIV=3.)
```

The MIN, MAX, and DIVISIONS parameters define the respective minimum, maximum and labeling increment values for the X and Y axes, when linear or logorithmic scales are used. Defaults are such that "good" scales are generated.

```
XMIN=bound >XMIN=20
```

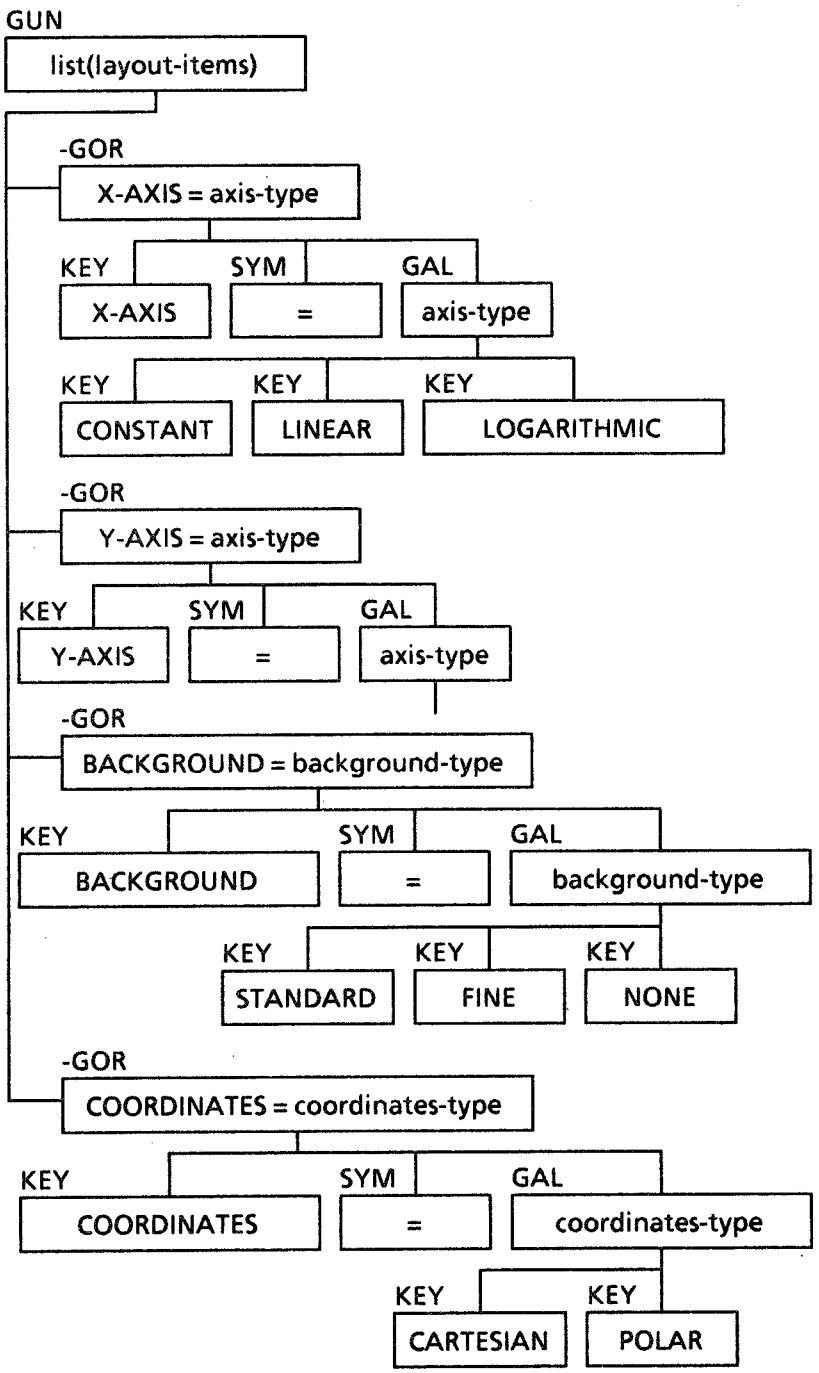
```
YMAX=bound >YMAX='5100'
```

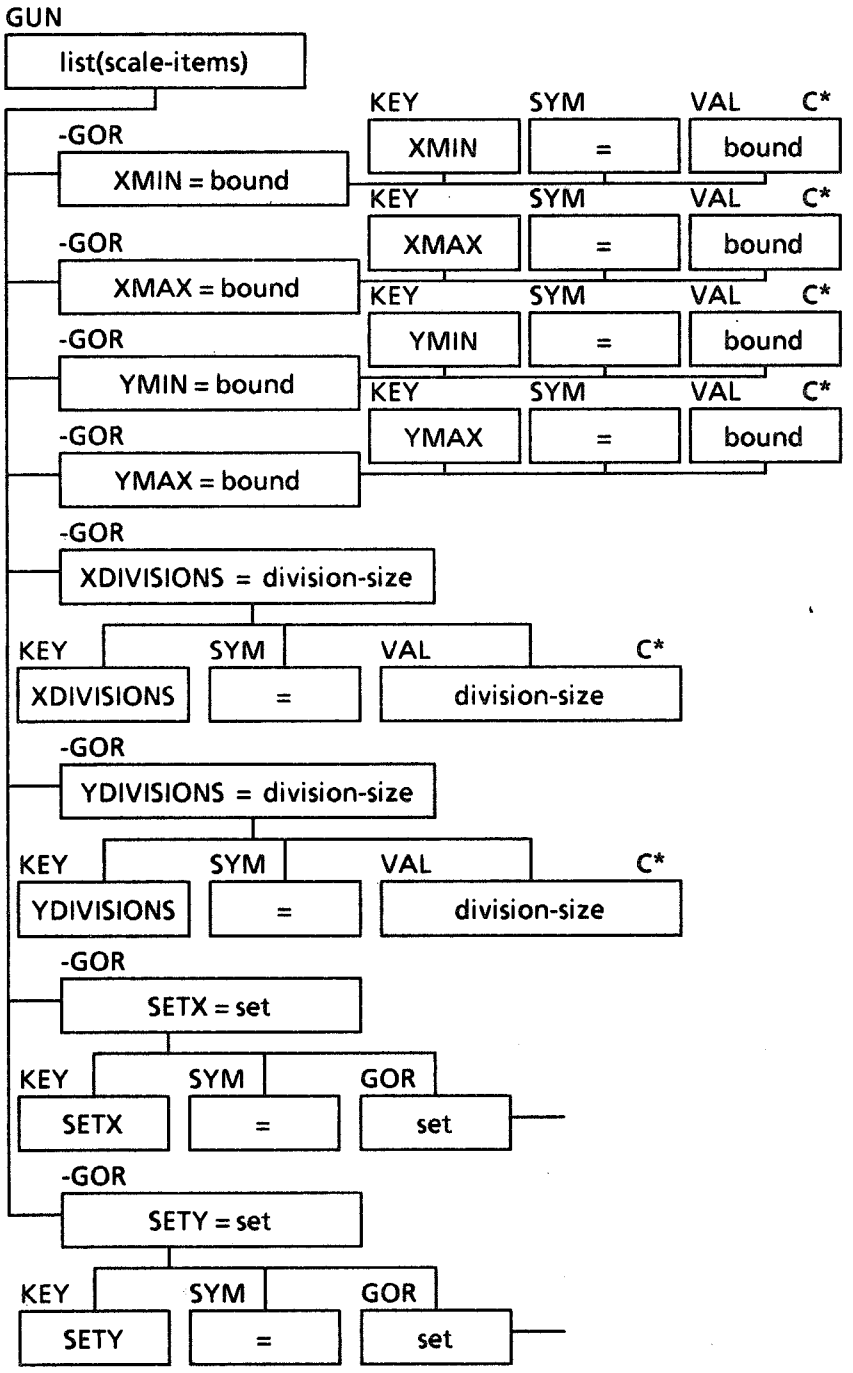
```
XDIVISIONS=division-size >XDIVISIONS=5.
```

The SETX and SETY parameters define X and Y axis information for CONSTANT scales. The set value specifies a set of point numbers, which may be used to define axis lower and upper limits and/or to skip axis labels on intermediate points (see Appendix B).

```
SETX=set >SETX=20..100
```

```
SETX=set >SETX=2..200&&2
```





PLOT-statement cont'd

PRINT allows you to print values from a relation or array, optionally interspersed with constants, to the currently defined OUTPUT file (see OUTPUT option in the SET IO command).

The "ra-spec" node gives the spec of the relation or array to be printed. A "$S" response as usual indicates that the spec is the same as the ra-spec most recently used.

The "print-names" node gives the list of items to be printed. The list may include index names, scalar or nonscalar attribute references, strings and other types of constants.

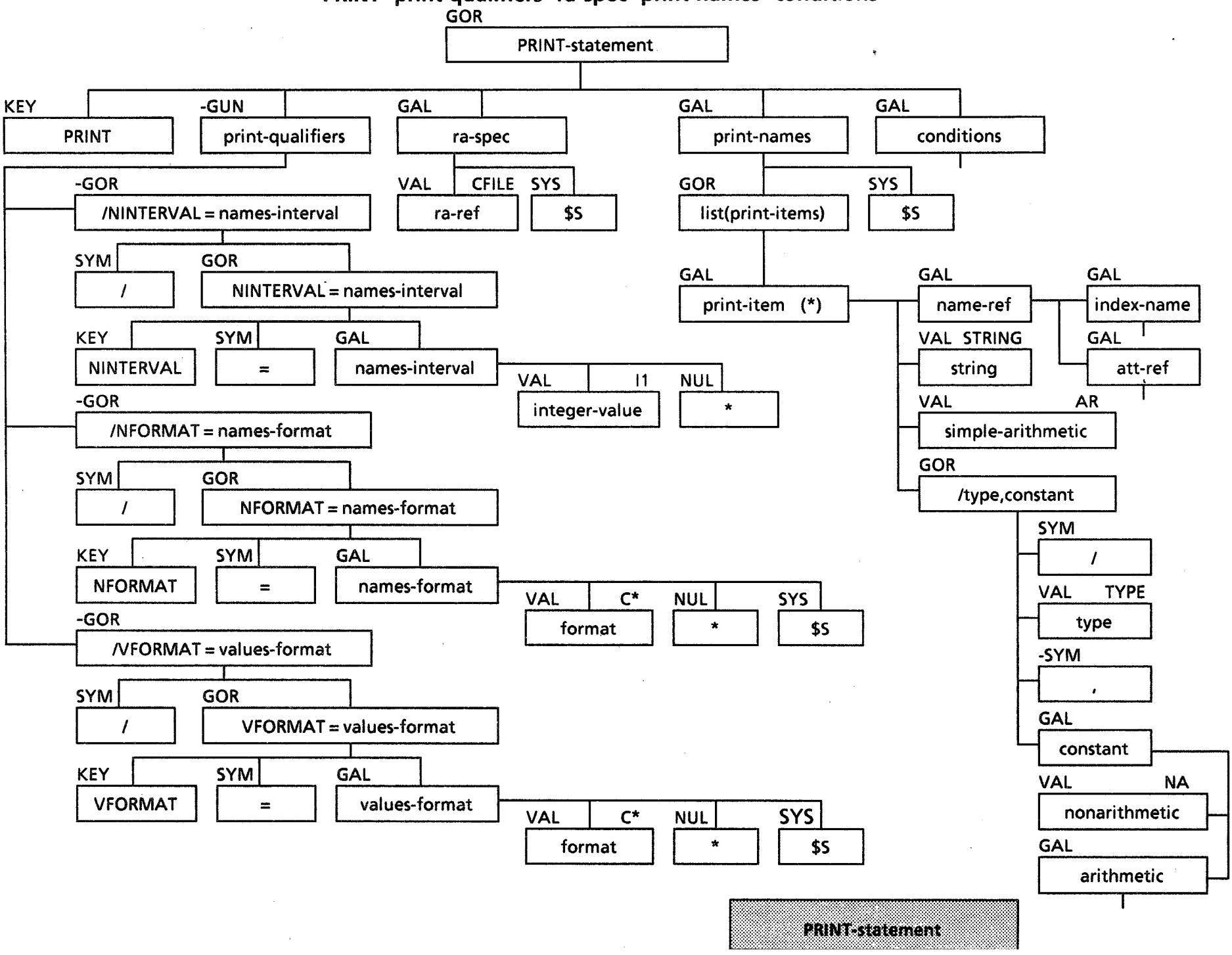
The "print-qualifiers" provide several formatting options. The NINTERVAL parameter controls the interval at which attribute name headings are interspersed with the output tuples; e.g. the value 10 causes a heading at every 10th tuple, and 0 causes a heading only at the beginning and end of all tuples. Default is no headings.

The NFORMAT parameter allows you to give a FORTRAN format by which all selected names in the headings are to be printed. Default format is a catenation of "IX,A10" for each name.

VFORMAT allows you to give a FORTRAN format by which all selected print-item values are to be printed. Default is a catenation of the individual attribute print formats stored in the data file.

Note that you may need to include appropriate slashes "/" in the names or values format, to prevent output buffer overflow in cases where a large number of columns are being printed.

```
ACE >PRINT/NI=0 X.DAT $A *

ACE >PRINT XYZ.DAT X,Y Z GT 100.

ra-spec >TEMP. ARRAY

ra-spec >$S

print-items >NODE,X,Y

print-items >'GRID',NODE,X,Y

print-items >'TIME=', TIME, TEMP.

print-qualifiers >/NI=10

print-qualifiers >/NI=0

print-qualifiers >/NI=-10


print-qualifiers >/NI=10+
/NF='A5, 3X, A10, 30X, ''***'', A10'


print-qualifiers >/NI=20+
/NF='I5, 3X, 4F10.3, 3X, D10.2'

values-format >'10F12.3/1X,5F10.3'
```


PRINT print-qualifiers ra-spec print-names conditions
GOR
PRINT-statement
KEY
PRINT
-GUN
print-qualifiers
GAL
ra-spec
GAL
print-names
GAL
conditions
-GOR
/NINTERVAL = names-interval
VAL
CFILE
SYS
ra-ref
$S
GOR
list(print-items)
SYS
$S
SYM
/
GOR
NINTERVAL = names-interval
KEY
NINTERVAL
SYM
=
GAL
names-interval
VAL
I1
NUL
integer-value
*
-GOR
/NFORMAT = names-format
SYM
/
GOR
NFORMAT = names-format
KEY
NFORMAT
SYM
=
GAL
names-format
VAL
C*
NUL
SYS
format
*
$S
-GOR
/VFORMAT = values-format
SYM
/
GOR
VFORMAT = values-format
KEY
VFORMAT
SYM
=
GAL
values-format
VAL
C*
NUL
SYS
format
*
$S
GAL
print-item (*)
GAL
name-ref
GAL
index-name
VAL STRING
string
GAL
att-ref
VAL
AR
simple-arithmetic
GOR
/type,constant
SYM
/
VAL
TYPE
type
-SYM
,
GAL
constant
VAL
NA
nonarithmetic
GAL
arithmetic
PRINT-statement

PROJECT projects an existing relation or array into a corresponding new relation or array. The new file contains a subset of selected attributes and tuples. If you give the new file spec the same as the old, a new version is generated. If you select all attributes and tuples, a copy operation results.

The "project-names" node allows you to select a variety of different index and attribute related entities to be projected into the new file. Existing attribute names are retained for use in the new file, unless you elect to redefine them via the "=new-att-name" node. Note that this node is required whenever you use the "index-name" or "att-function-ref" node. Formats revert to standard default values, unless you specify them via the "/att-format-redef" node. Note that a "/*" value for this node requests the existing format to be retained.

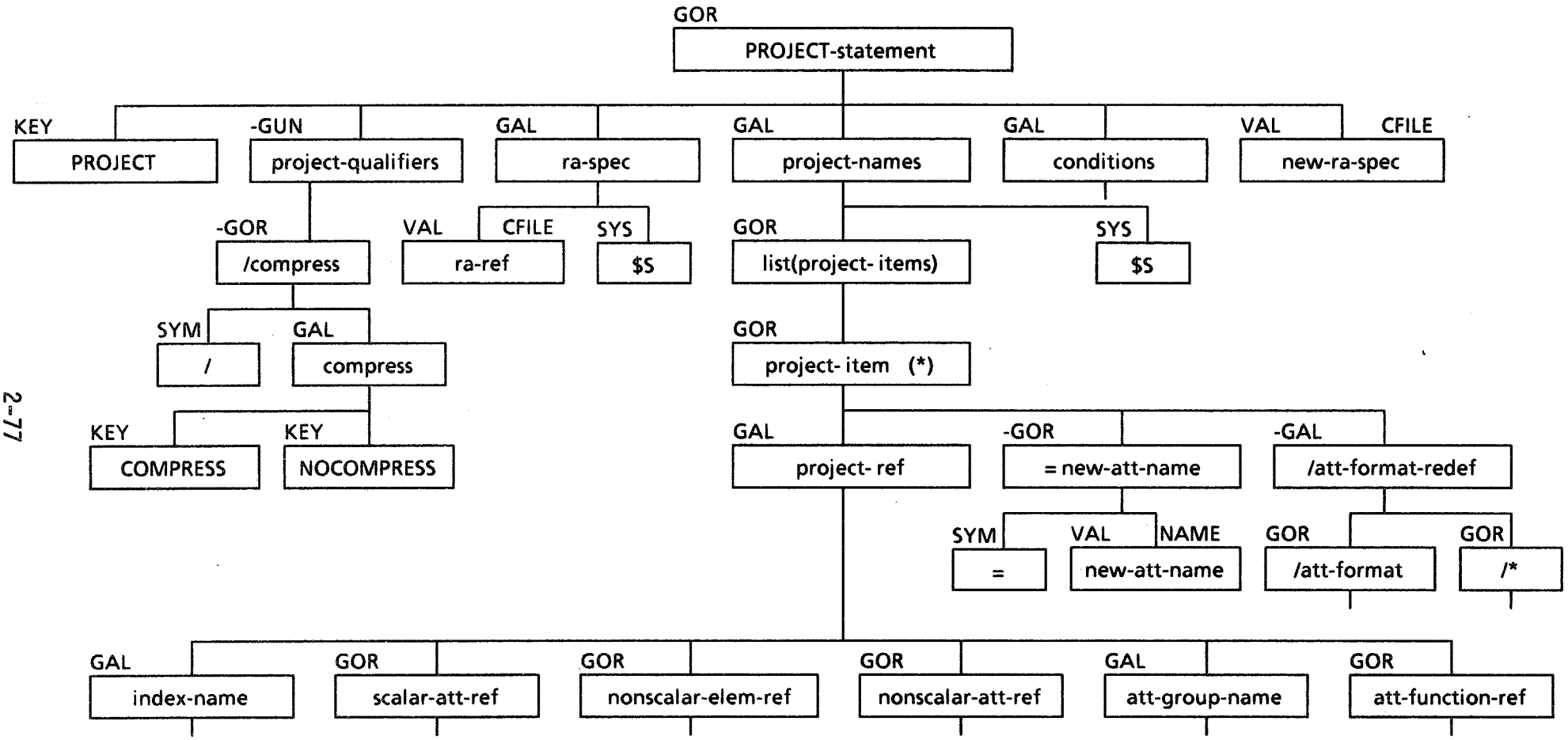
For an array, you may select entities associated with some or all indexes. Note that at least one entity must be selected associated with array index 0.

The compress option pertains to arrays. It allows you to delete in the new array any index whose size is unity. Compress deletes such indexes, and NOCOMPRESS (default) does not.

Note that for an array, the "conditions" node selects the set of rows, columns, etc. to be projected. Tuples not directly selected by the conditions may be projected, e.g. if they are in the same row or column as a selected tuple.

```
ACE >PROJECT A.DAT $A $I IN(101..200) B.DAT

ACE >PROJECT A.DAT A,C,D * X.NEW

ACE >PROJECT A.DAT $A * A.DAT

project-names >(A,B,$I=NDEX,C,D)

project-names >$I=I,A,X[..],B6[4..6]=B3

project-names >M,XYZ,X=A,Y/'F10.3'

project-names >A/*,B/*,$I=NDEX/'I5',C/*,D/*

project-item  >TXT[..]

project-item  >TXT[..]=TXT/'/1H ,A'

project-item  $L(TXT)=TXT_LENGTH/'I10'

project-names >$A1,$A2,TEMP

project-names >$A0,NODE,X,Y

ACE >PROJ/COMP A3.DAT $A0,A1,$A3 +
$I2 EQ 5  A2.DAT
```

PROJECT project-qualifiers ra-spec project-names conditions new-ra-spec



PROJECT-statement

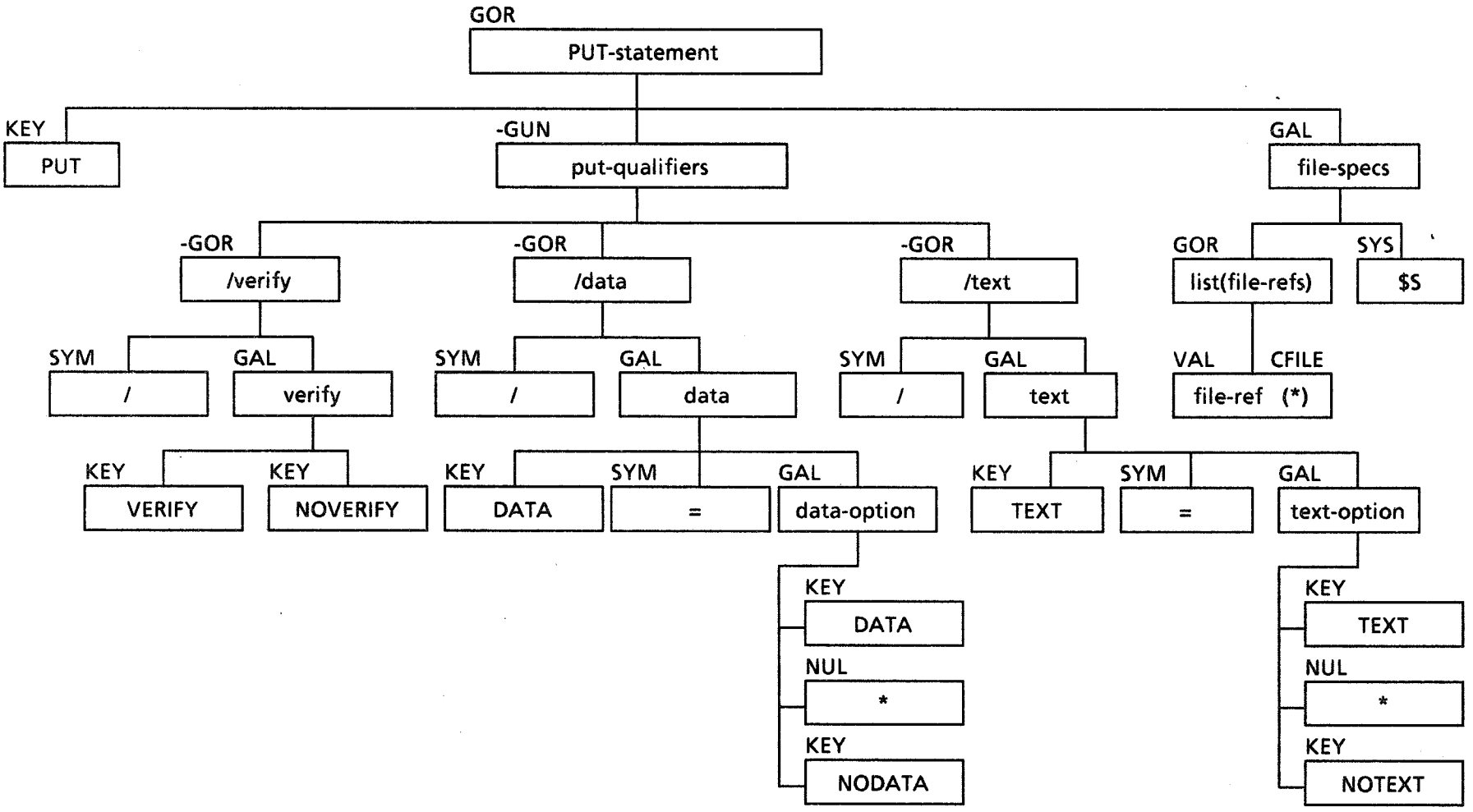
PUT allows you to transfer IAC cataloged files from the ACE workspace to a database. The verify option (default NOVERIFY) lets you confirm results of the transfer.

An IAC cataloged file may consist of a data and/or a text file. Note that the host default directory is considered to be a logical extension of the "workspace", and is the area from which a text file or an unstructured data file is transferred. The data-option and text-opion qualifiers let you specify whether you want the data and/or text file, respectively, to be transferred. Defaults are DATA=* and TEXT=NOTEXT. The value DATA indicates that the data file is to be transferred, and NODATA does not; * is like DATA, except it indicates that a new specification may be automatically generated for the data file in case there is a collision with an identical existing data file specification. The text-option values are similar to those for data-option.

Following the PUT command, you may want to use LIST/D to list files in the database, or GETC to obtain the database catalog for query of overall file attributes.

```
ACE >PUT A.B,X.DAT
*INFORMATION* Last ACE command processed

ACE >PUT/V X.Y
*INFORMATION* PUT of workspace X:1.Y;1
                to database X:1.Y;1 completed
*INFORMATION* Last ACE command processed

ACE >PUT/T=T PROB3.TEMP

ACE >PUT/T=* X.Y

ACE >PUT/V/D=D/T=T A.DAT

ACE >PUT/D=N/T=* (*.DAT,*.REL)
```

PUT put-qualifiers file-specs



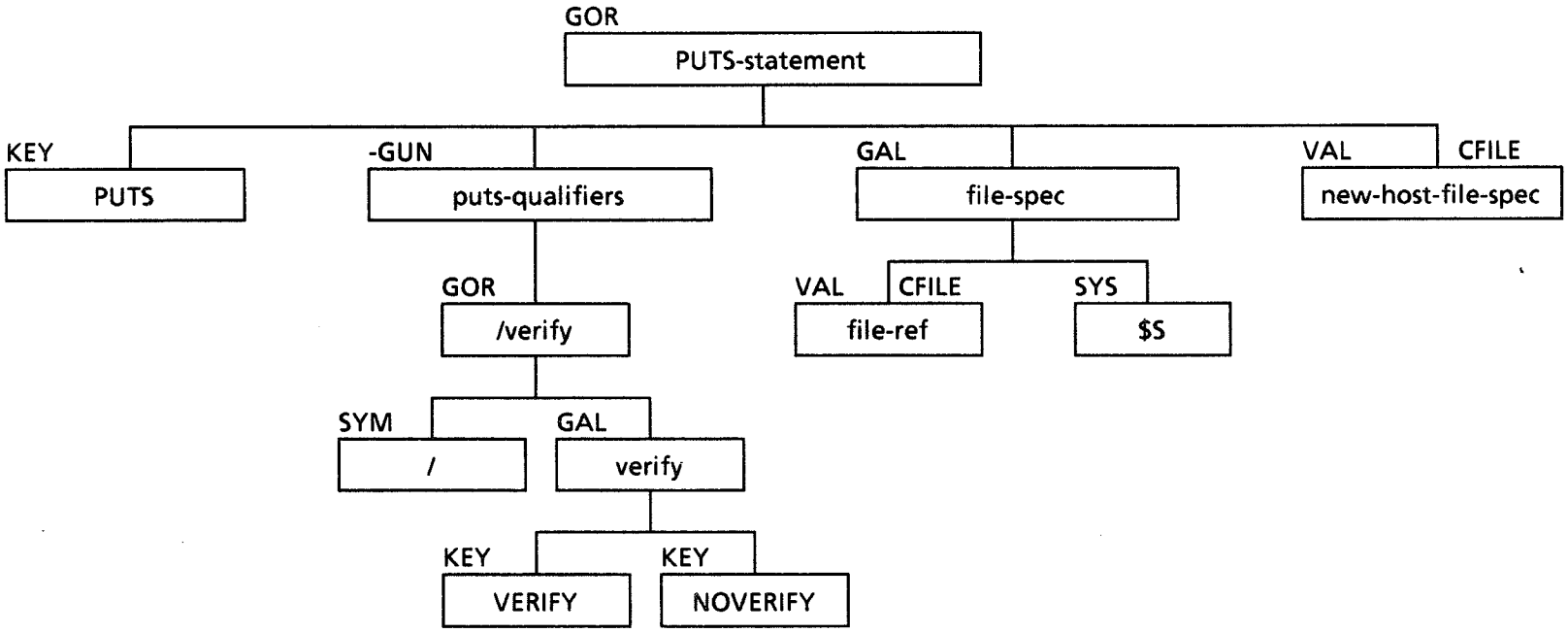
PUTS allows you to transfer an IAC structured data file (i.e. a relation or array) from the ACE workspace to a host directory. The verify option (default NOVERIFY) lets you confirm the transfer.

Note that the PUTS command requires two file specs. The first is the spec of the IAC cataloged file in the workspace. The second is the new host file spec, by which the file is to be identified in the host directory, and may include node and directory specifications.

Following the PUTS command, you may want to use HOST to list files in the host directory.

```
ACE > PUTS A.DAT A.DAT
*INFORMATION* Last ACE command processed

ACE > PUTS/V X.Y X.Y
*INFORMATION* PUTS of workspace X:1.Y;1
                  to hostfile DB1:[TLC]X.Y;1 ...
*INFORMATION* Last ACE command processed

ACE > PUTS Y.DAT DB2:[MAIN.MODEL2]X.DAT

ACE >PUTS X:1.DAT A.DAT
```

PUTS puts-qualifiers file-spec new-host-file-spec



PUTS-statement

REDEFINE allows you to redefine certain properties of selected attributes within a workspace relation or array file. The example shown changes an attribute name (from NODE to GRID) and the attribute print format (to I9).

The list (att-redefs) node specifies a list of attribute name,type/format triplets to be redefined. Any attribute not redefined remains as before, and an * for any item in a triplet indicates no change for that item. The format item is optional within a triplet; if the format is not given, a default format is assigned.

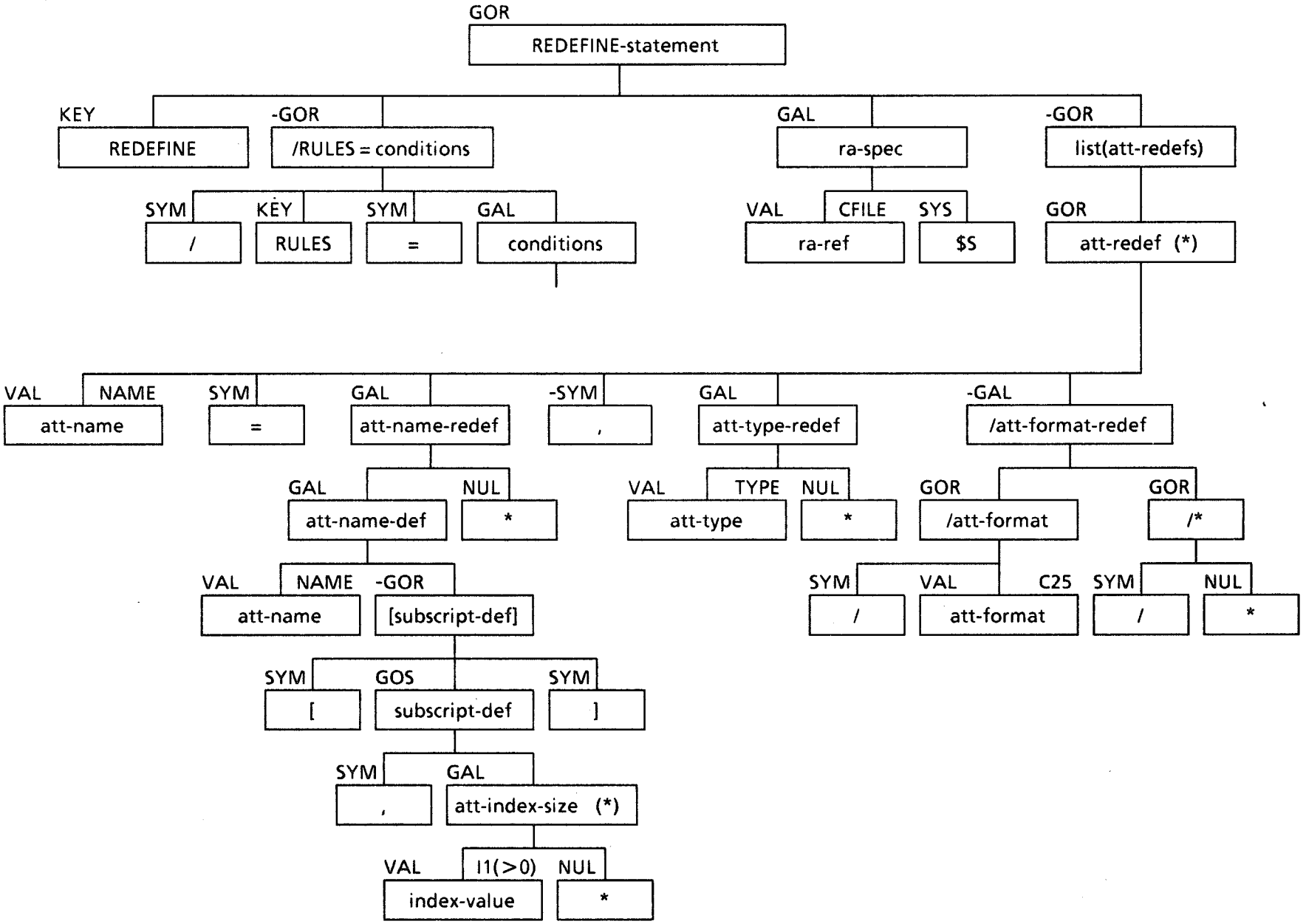
For a relation which is empty, i.e. contains no tuples, you may redefine any of the attribute characteristics (name, subscript, type, format).

For an array, or a non-empty relation, you may redefine the name and format. Also the subscript of a nonscalar attribute may be redefined, provided that the number of elements per attribute is not changed. This provision requires that a) the product of the fixed index sizes remains constant; and b) the number of variable indexes remains constant. Note that correlation between old and new variable indexes is based on their corresponding orders. As examples, a subscript may be redefined from [6] to [3,2], from [*,6] to [3,*,2], from [1,*] to [*], or vice versa.

Following the REDEFINE command, you may want to use SHOW FILE to show file characteristics.

```
ACE >REDEF NODES.DAT NODE=GRID,*/I9

list(att-redefs) >(TX=TI,* TY=T2,*)
list(att-redefs) >(TEMP=*,*/'3X,D20.10')
list(att-redefs) >(X=*,R2/* Y=*,R2/*)

list(att-redefs) >(STRESS=S3[3], R1/F10.3)
list(att-redefs) >T2=T3[*,3,*],R2 A=*,C10)
list(att-redefs) >(CHAR=*,C*/'''TEXT='',A')
list(att-redefs) >(STRESS=S3[3],*/F10.3)
list(att-redefs) >T2=T3[*,3,*],* A=*,*)
list(att-redefs) >CHAR=*,*/'''TEXT='',A')
```

REDEFINE /RULES = conditions ra-spec list (att-redefs)

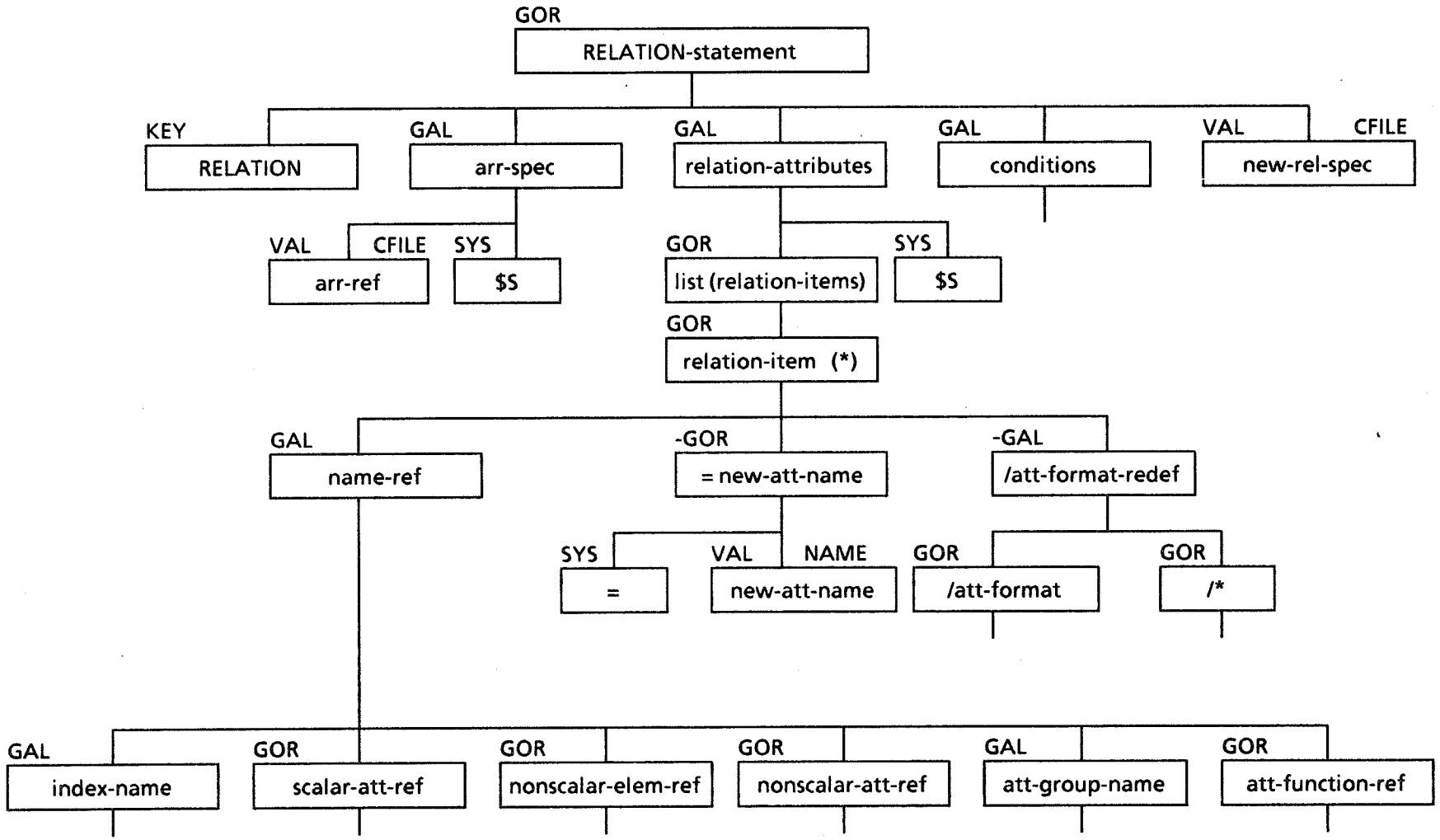


RELATION uses selected attributes and tuples from an array, to create a new relation. The operation is similar to that of the PRINT command applied to an array, except that the selected values are stored into a relation. (See also the VECTORS command, for a different array-to-relation transformation capability).

The "relation-attributes" node allows you to select a variety of different index and attribute related entities to be included in the new relation. Existing array attribute names are retained for use in the new relation, unless you elect to redefine them via the "=new-att-name" node. Note that this node is required whenever you use the "index-name" or "att-function-ref" node. Formats revert to standard default values, unless you specify them via the "/att-format-redef" node. Note that a "/*" value for this node requests the existing format to be retained.


```
ACE >RELATION X.ARR A,B,C   A NE 0  X.REL
ACE >REL A.DAT $A1, TEMP  TIME ME 2  A.TEMP
ACE >REL A.DAT $A * A.REL

relation-attributes >(A,B,$IO=NDEX,C,D)
relation-attributes >$I1=I,A,X[..],B6[4..]=B3
relation-attributes >M,XYZ,X=A,Y/'F10.3'
relation-attributes >A/*,B/*,$I2=NDEX/'I5'
relation-item >XLABEL[..]
relation-item >XLABEL[..]=XLABEL/'/1H ,A'
relation-item >$REAL(ZVAL)=ZREAL/'E12.4'
relation-item >$A/*
```

RELATION arr-spec relation-attributes conditions new-rel-spec



RELOAD allows you to reload values in an existing relation or array file. After the RELOAD command syntax is processed, ACE will give you interactive assistance while you input values. The $ symbol terminates reloading.

The reload-names and the tuples selected by the conditions determine the set of values that can be reloaded. You can think of this as an "unmasking" operation - only the unmasked set of values can be reloaded or accessed in any way. The reload-names specify the scalar attributes and/or nonscalar attribute elements which are accessible.

The prompt names specify attribute values which are to be provided to you as prompts prior to reloading of the next value. You can use these to determine where you are and whether you want to reload a particular value.

You insert the new values in data input lines following the RELOAD command line.

A data line with at most blank(s), causes the current attribute position to be skipped, while a data line ending in a dollar sign (i.e. $) causes reloading to terminate. If your input line begins with a percent (i.e. %), then the line is a control line. Control lines of %H and %E respectively display help information and examples. Use %HE to get additional information and examples, on absolute and/or relative position control during reloading.

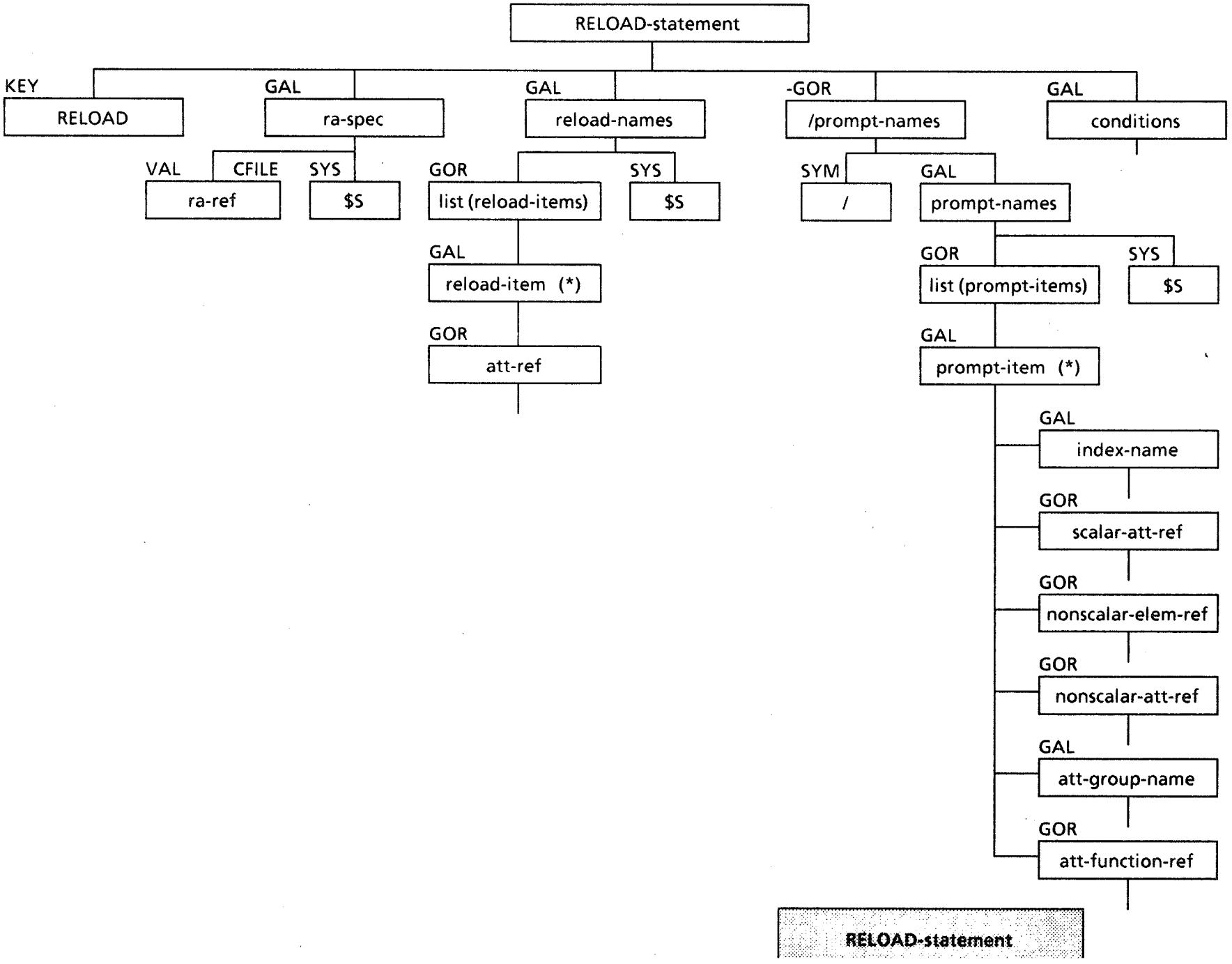
Following the RELOAD command, you may want to use SHOW FILE to show data file structure and organization, or commands such as PRINT to determine detailed characteristics.

```
ACE >RELOAD A.DAT X $I2 EQ 3

ACE >RELOAD A/DAT X,Y/$I1 *

ACE >RELOAD NODES.REL Z/$I Z GT 5
```

```
ACE >RELOAD A.DAT X,Y/$I1 $I1 GT 10
$I1
      11
X >
```

```
X >1.5 6.3 5.1
Y >

X >%H

X >6.0,7.3,10.5 $

X >T3 A2

X >101 Y[3]
```

RELOAD ra-spec reload-names /prompt-names conditions



RELOAD-statement

RUN executes one of the IAC provided modules (see Section 3). The command includes the module name, and a list of run parameters. A parameter is of the form keyname=keyvalue or keyname=list(keyvalues). The dashed line node boxes in the diagram indicate that the enclosed prompt names are only typical values, since they depend upon the particular module being executed.

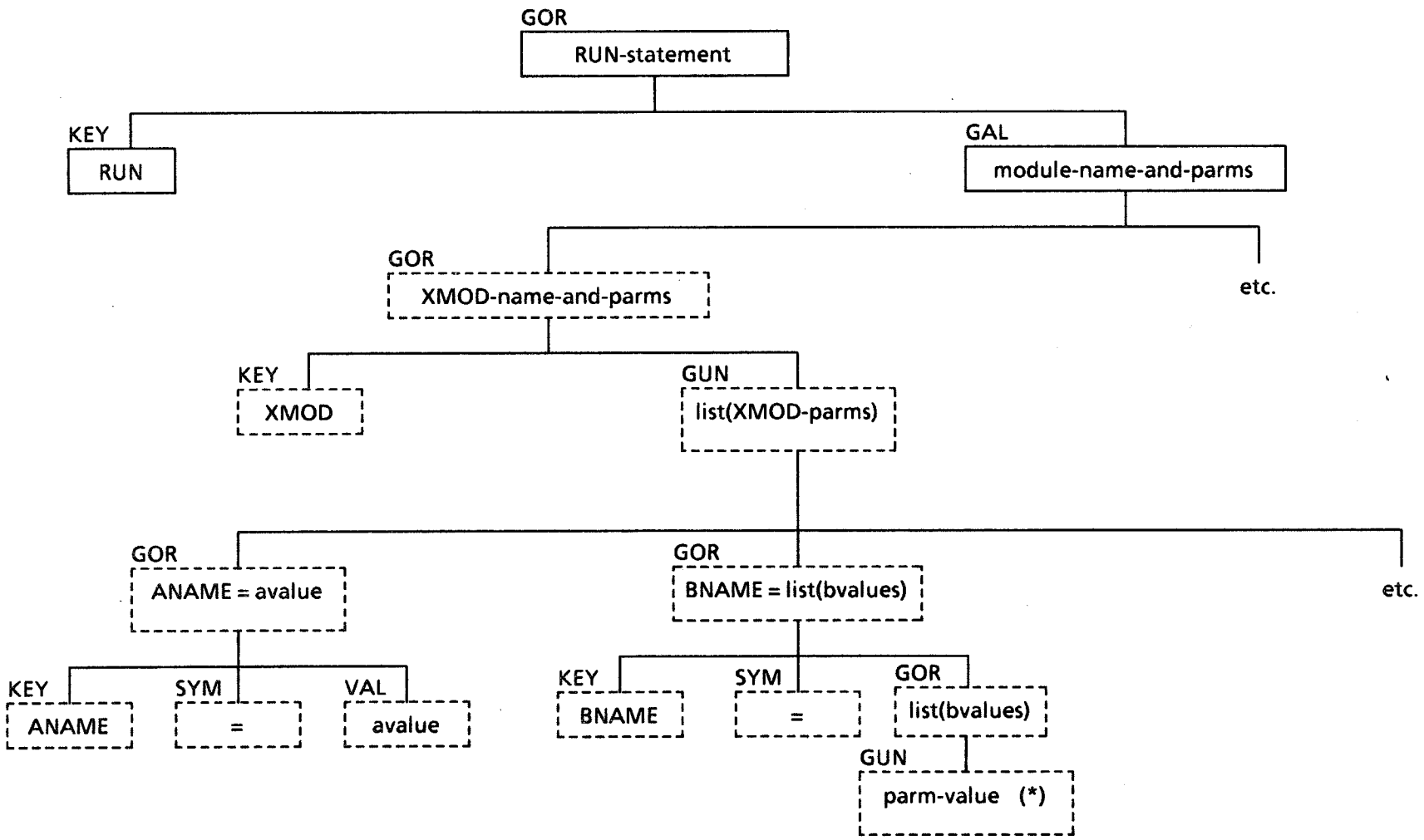
The module name, e.g. XMOD or NASTRAN, is the name by which the module is known within the IAC system. If you do not remember the details for executing a particular module, a convenient method of operation is the following.

1) Type "RUN", followed by a carriage return.
2) In response to the "module-name--and-parms" prompt, type "%HE". This will give you a list of available modules, with other help and examples information.
3) Type the desired module name, followed by a carriage return.
4) In response to the "list(...-parms)" prompt, again type "%HE". This will give you a list of available run parameters for the module, with help and examples.
5) Accept the auto-input left parenthesis which ACE provides you, by typing a space followed by a carriage return.
6) As ACE prompts you for each run parameter, request help and examples via "%HE". Then either input the value(s) for that parameter, or reject (if an optional parameter) by a carriage return.
7) Follow the last input parameter by a right parenthesis.

Note that you will usually want to ignore any trivial prompts, e.g. for a comma, by hitting a carriage return.


```
ACE >RUN NASTRAN (F=STATION 2, PRT=YES)

ACE >RUN USER (EXE=MYTEST)


ACE >RUN
module-name-and-parms >NASTRAN
list )NASTRAN-parms) >(F=TRUSS)

module-name-and-parms >%HE

The following modules are avialable.
•
•
•

list(FEMNET-parms) >(%HE
•
•
•

RUN NASTRAN (
F= >F=%HE
•
•
•
```

RUN module-name-and-parms



SAMPLE provides information to determine whether two groups of data can be combined as part of the same normal population. Each data group can be selected from a relation or an array. Suffixes 1 and 2 in the command diagram denote nodes associated with the two respective groups.

Summary statistics are computed and printed for each group. Then the F-test measures difference in variabilities, and the T-test measures diference in means. F and T significance levels are printed, which you should compare to a specified rejection level. See the upper level help information for more details.

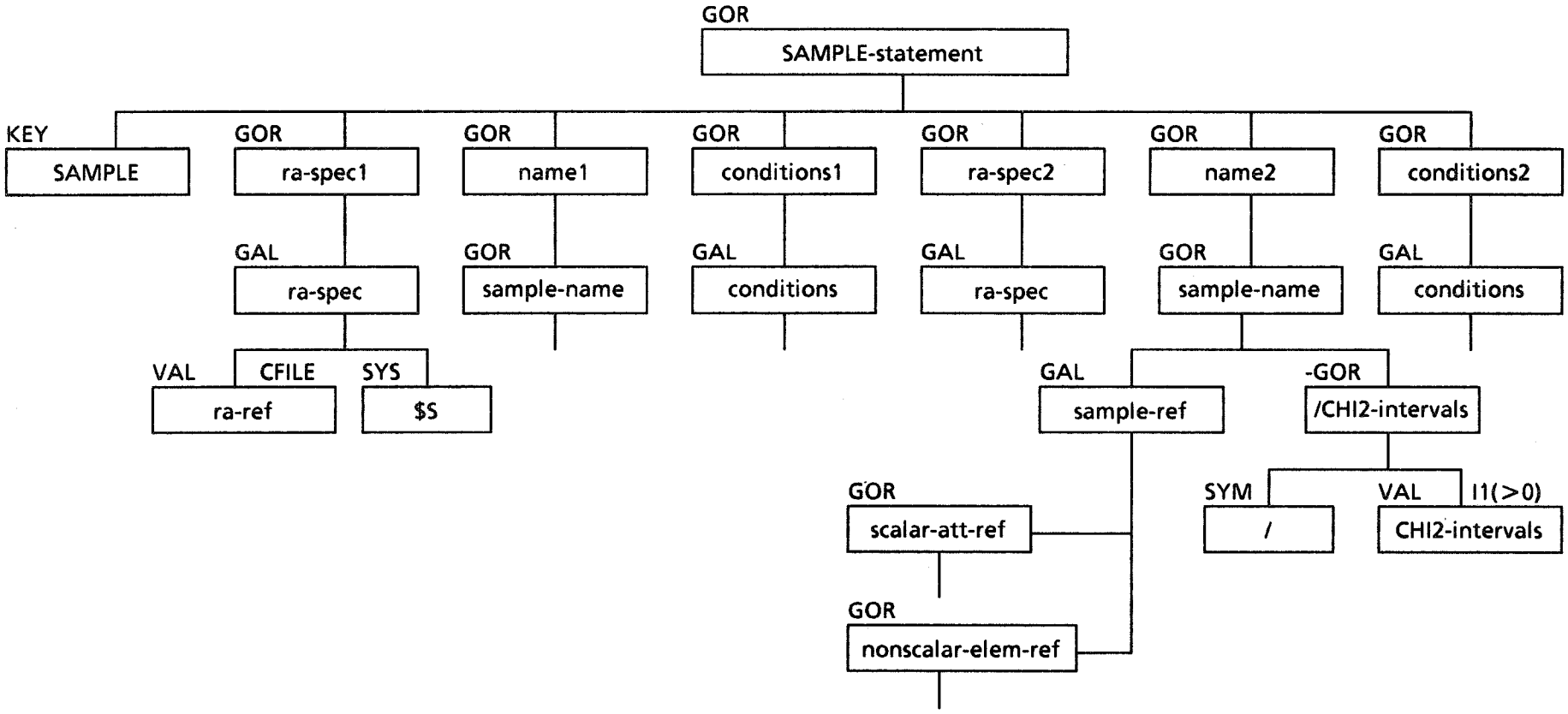
The "name 1" or "name 2" node may be either a scalar attribute or an element of a nonscalar.

Computations are based on the statistical CHI-square value. The "/CHI2-intervals" node defines the number of intervals to be used for computing CHI-square. Default number of intervals is the square root of the number of tuples selected by the conditions.

Note that results of the SAMPLE command may be strongly dependent upon the "CHI2-intervals" value. The default is generally recommended.

```
ACE >SAMPLE A.TEST X * B.TEST X *

ACE >SAMPLE A.ARR A  $I2 EQ 3 +
$S A  $I2 EQ 4

ACE >SAMPLE
ra-spec1 >%UHE
•
•
•

name1 >YIELD

name1 >STRESS[2]

/CHI2-intervals >/10
```

SAMPLE ra-spec1 name1 conditions1 ra-spec2 name2 conditions2



SAMPLE-statement

SET IO sets ACE IO file specifications. These specifications may be cleared (set to standard values) via the CLEAR IO command, and displayed via the SHOW IO command. See CLEAR IO for definition of the standard values.

```
ACE >SET IO (IN=X.DAT, OUT=Y.DAT)
ACE >SET IO I=SUPERTAB.GEN
ACE >SET IO (JOUR='', MESS='*')
```

The "INPUT=input-file" node defines the primary input file or device. The input-file value may be a host file spec, the asterisk string '*', or a reference to an existing input file definition (e.g. $AMACRO).

```
input-file >PROC.DAT

input-file >'*'
```

The "AMACRO=input-file" node defines the input file or device from which AMACRO responses are received. The input-file value may be a host file spec, the asterisk string'*', the blank string ' ', or a reference to an existing input file definition (e.g. $INPUT).

```
input-file >'*'

input-file >$INPUT
```

The "OUTPUT=output-file" node defines the file or device used for major computational and query output. The output-file value may be a host file spec, the asterisk string '*', the blank string ' ', or a reference to an existing output file definition (e.g. $MESSAGE).

```
output-file >MODDATA.DAT

output-file >'*'
```

The "PROMPT=output-file" node defines the file or device used for recording prompts and auto-input. The output-file value may be a host file spec, the asterisk string '*', the blank string ' ', or a reference to an existing output file definition (e.g. $MESSAGE).

```
output-file >PR.DAT

output-file >'*'
```

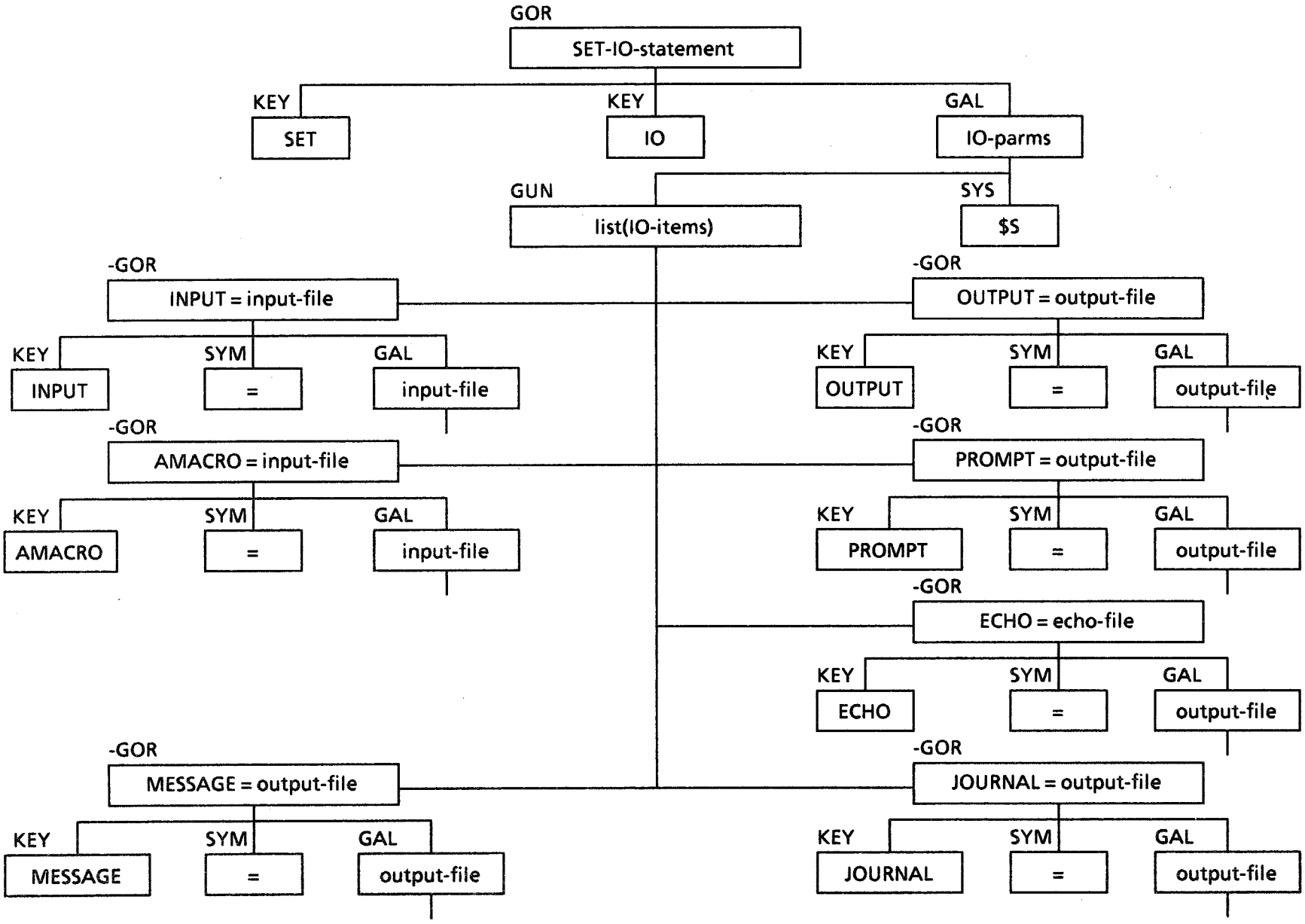
The "MESSAGE=output-file" node defines the file or device used for IAC system messages (information, warning and error). The output-file value may be a host file spec, the asterisk string'*', the blank string '', or a reference to an existing output file definition (e.g. $OUTPUT).

```
output-file >MESS.OUT

output-file >$OUT
```

The "JOURNAL=output-file" node defines the file or device used for journaling. The output-file value may be a host file spec, the asterisk string '*', the blank string ' ', or a reference to an existing output file definition (e.g. $MESSAGE).

```
output-file >JOU.DAT

output-file >''
```

SET IO IO-parms



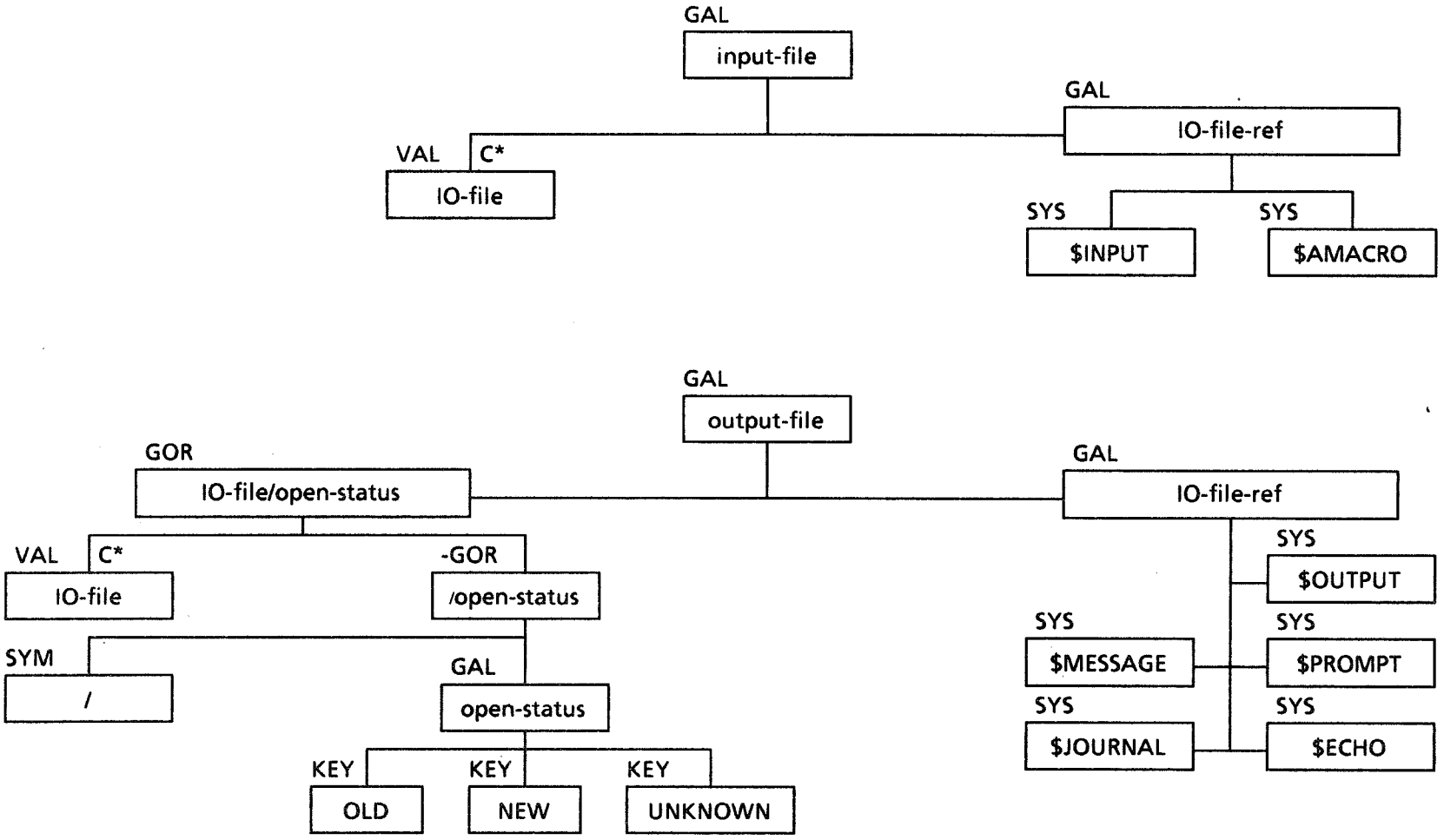
SET-IO-statement

The "input-file" node is used to define an ACE input file or device. It may have one of the following values.

host file spec	Identifies an existing file, from the beginning of which input is to be initiated.	input-file >A.DAT
'*'	Indicates a return to the interactive terminal or to the current position within the primary input file.	input-file>'*'
' '	Indicates the absence of the input operation.	input-file >' '
$ reference	Identifies a reference to an existing input definition to be used for defining the current input operation.	input-file >$INPUT

The "output-file" node is used to define an ACE output file or device. It may have one of the following values.

host file spec	Identifies a new or existing file, to which output is to be respectively written or appended. The spec may be followed by a "/" and one of the Keyvalues OLD, NEW (default) or UNKNOWN. OLD indicates appending, and UNKNOWN indicates appending if the file exists.	output-file >B.OUT
'*'	Indicates the interactive terminal or the batch log file.	output-file >'*'
' '	Indicates the absence of the output operation.	output-file >' '
$ reference	Identifies a reference to an existing output definition, to be used for defining the current output operation.	output-file >$OUTPUT


GAL
input-file
VAL
C*
IO-file
GAL
IO-file-ref
SYS
$INPUT
SYS
$AMACRO
GAL
output-file
GOR
IO-file/open-status
VAL
C*
IO-file
-GOR
/open-status
SYM
/
GAL
open-status
KEY
OLD
KEY
NEW
KEY
UNKNOWN
GAL
IO-file-ref
SYS
$OUTPUT
SYS
$MESSAGE
SYS
$PROMPT
SYS
$JOURNAL
SYS
$ECHO

SET MACRO sets ACE macro parameters. These parameters may be cleared (set to standard values) via the CLEAR MACRO command, and displayed via the SHOW MACRO command. See CLEAR MACRO for definition of the standard values.

```
ACE >SET MACRO S=OFF

ACE >SET MAC (REL=R.TXT, IO=ID2)
```

The "STATUS=status" node specifies whether macros are active (ON) or inactive (OFF). When macros are active, a single "@" symbol in the command input is interpreted as the start of a macro reference, and non-macro "@" symbols must be doubled.

```
status >OFF
```

The "RELATION=rel-spec" node gives the spec of the macro relation in the workspace. This relation is used replace macro references by corresponding macro values.

```
rel-spec >XY.DAT
```

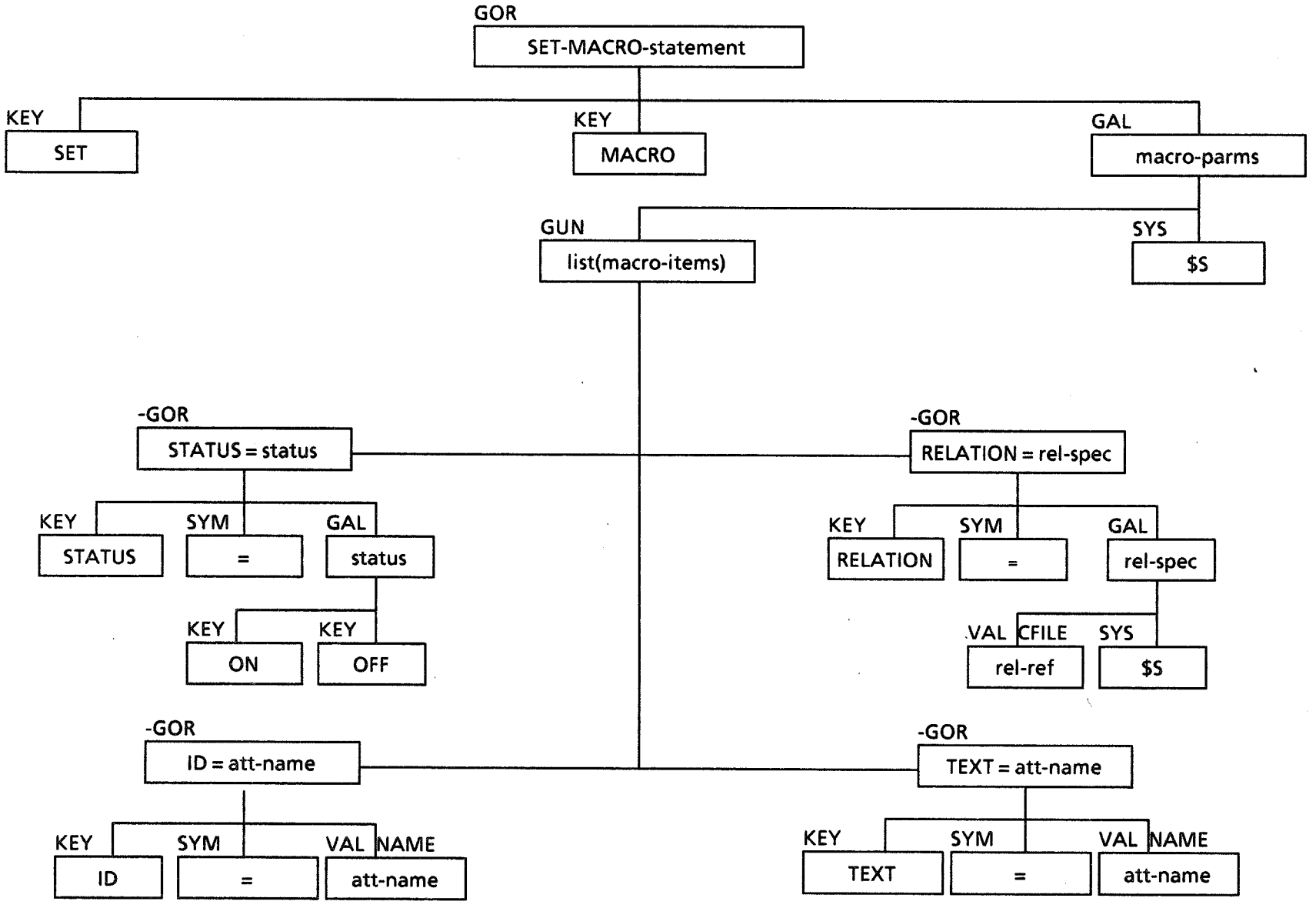
The "ID=att-name" node gives the name of the id attribute within the macro relation. The value of this attribute is matched with the macro reference string.

```
att-name >TEMPID
```

The "TEXT=att-name" node gives the name of the text attribute within the macro relation. The value of this attribute, taken from the tuple where the id matches the macro reference, becomes the macro replacement string.

```
att-name >TXT
```

SET MACRO macro-parms



SET-MACRO-statement

SET MESSAGE sets ACE message control parameters. These parameters may be cleared (set to standard values) via the CLEAR MESSAGE command, and displayed via the SHOW MESSAGE command. See CLEAR MESSAGE for definition of the standard values.

ACE >SET MESSAGE W=OFF

The "INFORMATION=status" node specifies whether information level messages are to be displayed. ON indicates that they are.

status >ON

The "WARNING=status" node specifies whether warning level messages are to be displayed. ON indicates that they are.

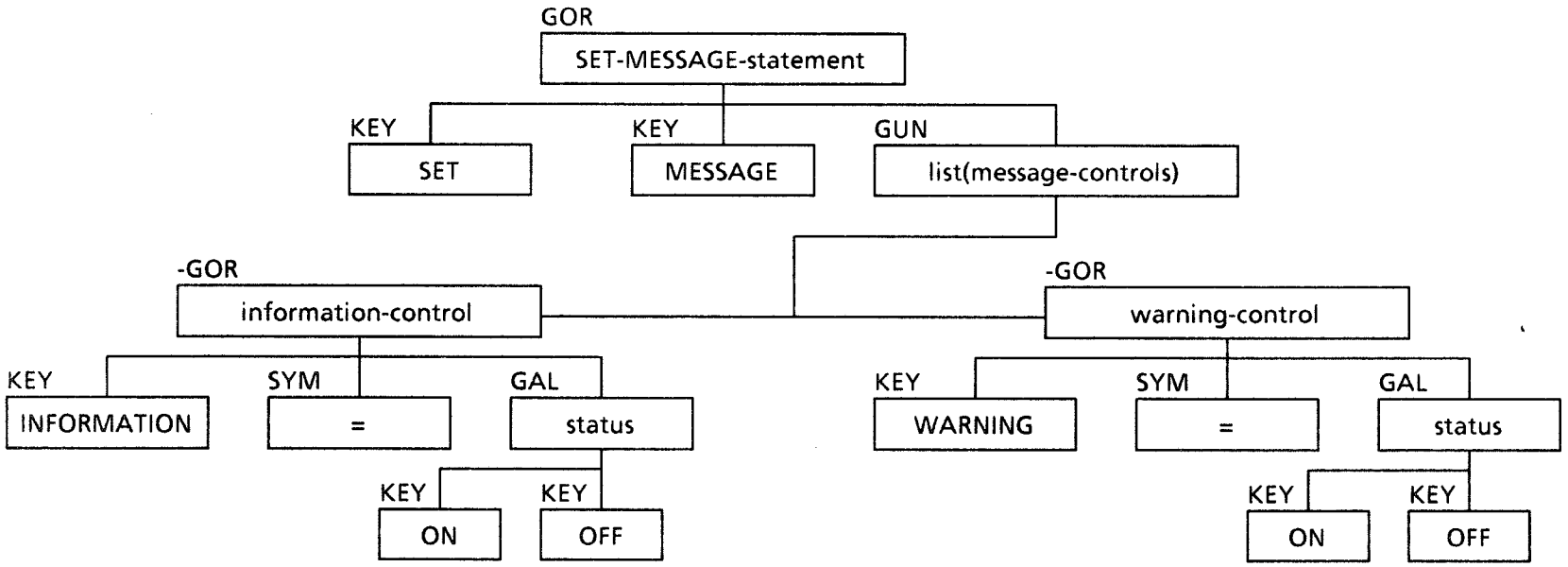
status >OFF

Note that error level messages are displayed in any case.

You may use the SET IO command to specify the output file onto which messages are to be written.

ACE >SET IO MESS=X.MESS

SET MESSAGE list(message-controls)



SET-MESSAGE-statement

SET UCATT sets user definable catalog attributes, for a specified list of files in the workspace or database. See the CATALOG command for associated information.

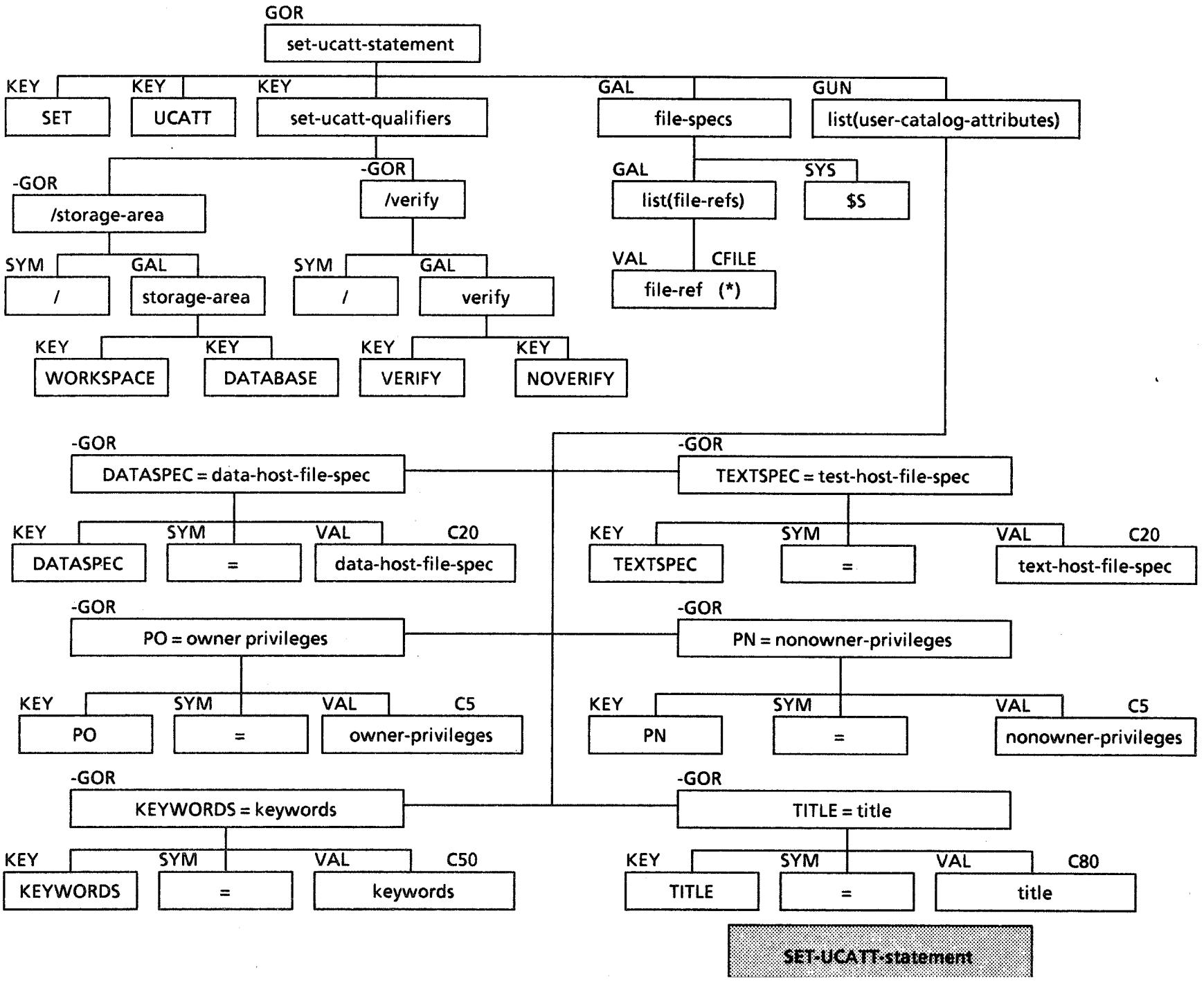
You may use qualifiers to specify the storage area and to request verification of the operation. Default qualifiers are WORKSPACE and NOVERIFY.

Each attribute value is a character constant, with maximum size as indicated in the diagram. In general a blank value indicates absence of data for the attribute (e.g. blank value for PN indicates no privileges for the nonowner). An asterisk value indicates that a database dependent default value is to be generated for insertion into the database catalog.

Following the SET UCATT command, you may want to use LIST/F to list files in the appropriate workspace or database, or GETC to obtain the database catalog for more detailed query.


```
ACE >SET UCATT/D A.DATA DATA=A.DAT

ACE >SET UC X.* (PO=RWD, PN=R)

set-ucatt-qualifiers >/D

set-ucatt-qualifiers >/V/D

data-host-file-spec >BIG.ARRAY

data-host-file-spec >'*'

nonowner-privileges >'*'

Keywords >'DESIGN_2 MOD_3 CONTROLS'
```

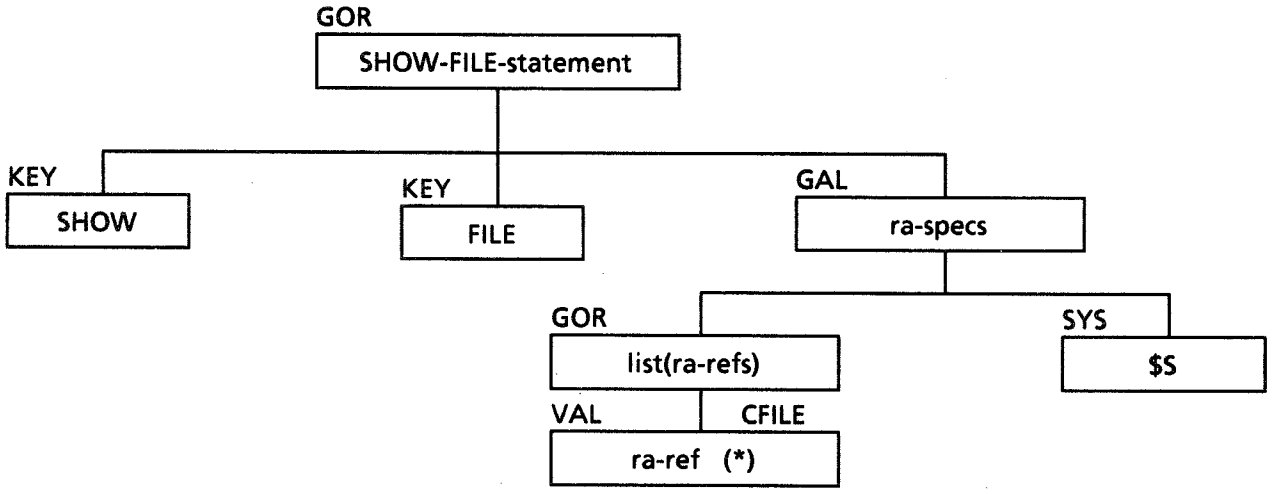
SET UCATT set-ucatt-qualifiers file-specs list(user-catalog-attributes)



SHOW FILE displays overall characteristics for a specified list of relation and/or array files resident in the workspace. Characteristics include number of tuples; array dimensions and structure; and attribute names, dimensions, types and print formats.

```
ACE >SHOW FILE *.*

ACE >SHOW FIL A.DAT,*.REL
```

SHOW FILE ra-specs

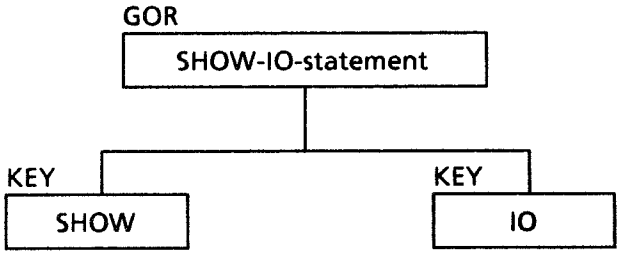


SHOW-FILE-statement

SHOW IO displays ACE IO file specifications. These specifications may be set via the SET IO command, and cleared (set to stnadard values) via the CLEAR IO command.

```
ACE >SHOW IO
```

SHOW IO

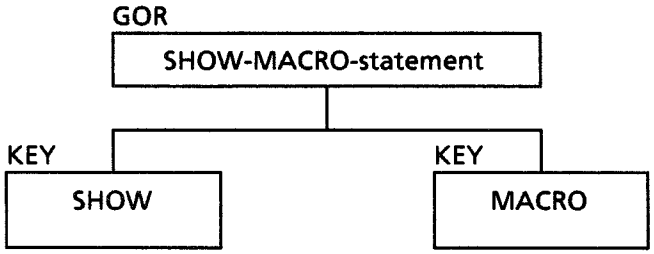


SHOW-IO-statement

SHOW MACRO displays ACE MACRO parameters. These parameters may be set via the SET MACRO command, and cleared (set to standard values) via the CLEAR MACRO command.

```
ACE >SHOW MACRO
```

SHOW MACRO

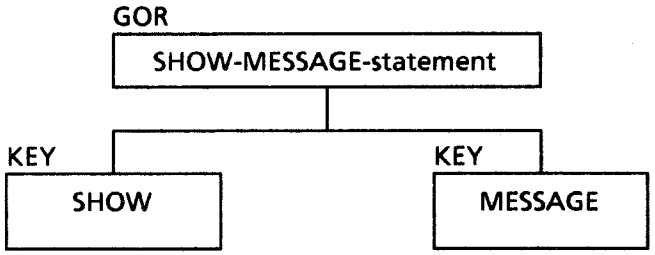


SHOW-MACRO-statement

SHOW MESSAGE display ACE message control parameters. These parameters may be set via the SET MESSAGE command, and cleared (set to standard values) via the CLEAR MESSAGE command.

ACE >SHOW MESSAGE

SHOW MESSAGE



SORT sorts the tuples of a relation, or sorts one non-zero index of an array, based on comparison of specified attribute(s).

```
ACE >SORT STUFF.DAT X
```

The "sort-qualifiers" node allows you to override the default options. A "sort-order" value of LOHI (default) sorts via increasing values, and HILO uses decreasing values. The "complex-sort-option" values control sorting via complex attributes; ZRI (default) causes sorting via real and then imaginary, ZIR via imaginary and real, ZMA via magnitude and angle, and ZAM via angle and magnitude.

```
sort-qualifiers >/HILO
sort-qualifiers >/ZMA/HILO
```

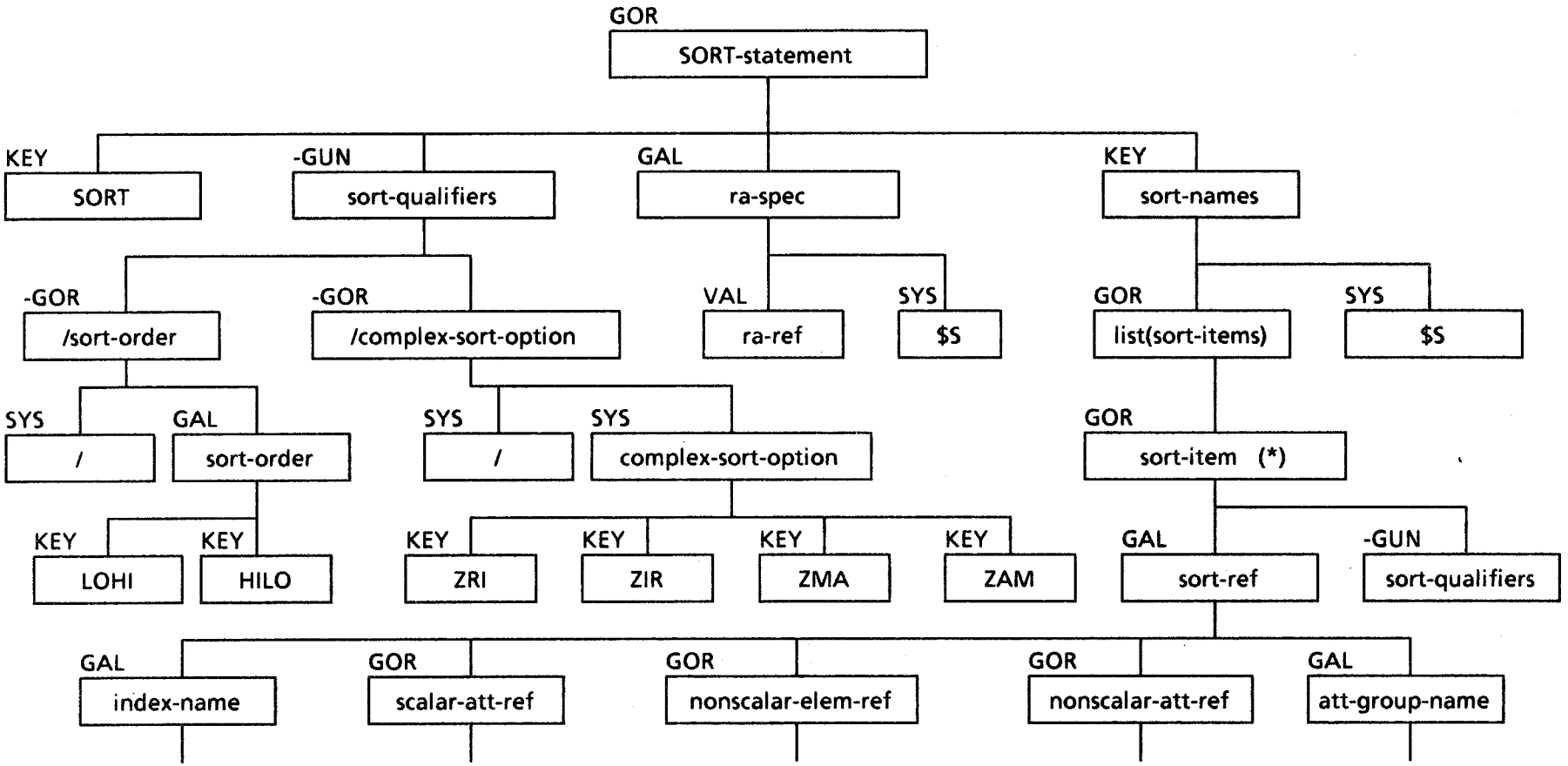
The "sort-names" node specifies the comparison attributes for sorting. If more than one is specified, the first one is the primary attribute; if two or more tuple positions are equivalent based upon the primary comparison, an attempt is made to resolve the position order based on the second specified attribute; etc. Note that if you want tuples with equivalent positions to remain in the same relative order, you can use the index name (e.g. $I) as the last name for comparison. Note that for an array, all names must be associated with the same non-zero index.

```
sort-names >X
sort-names >X,Y,Z
sort-names >A,B,$I
sort-names >$A2,$I2
sort-names >ZVAL/ZAM,X
```

Each sort name may be one of several kinds of scalar or nonscalar entities, as shown in the diagram.

```
sort-item >B
sort-item >A[41/80]
sort-item >AVEC[2]
sort-item >ABC[4..6]
```

SORT sort-qualifiers ra-spec sort-names



SORT-statement

SUBMIT initiates a batch ACE module run. The batch ACE module is given the task of executing one or more ACE commands, which were specified as input-string parameter(s) in the SUBMIT command.

```
ACE>SUBMIT (QUEUE='SYS$LO', PRT=YES)+
'RUN NASTRAN (F=PROB3)'

ACE>SUBMIT 'SET IO I=BATCH.JOB'
```

The "submit-parms" node allows you to specify the following batch control parameters, listed with their default values.

```
subm  it-parms >(NAME=JONES,LOG=X.DAT)
```

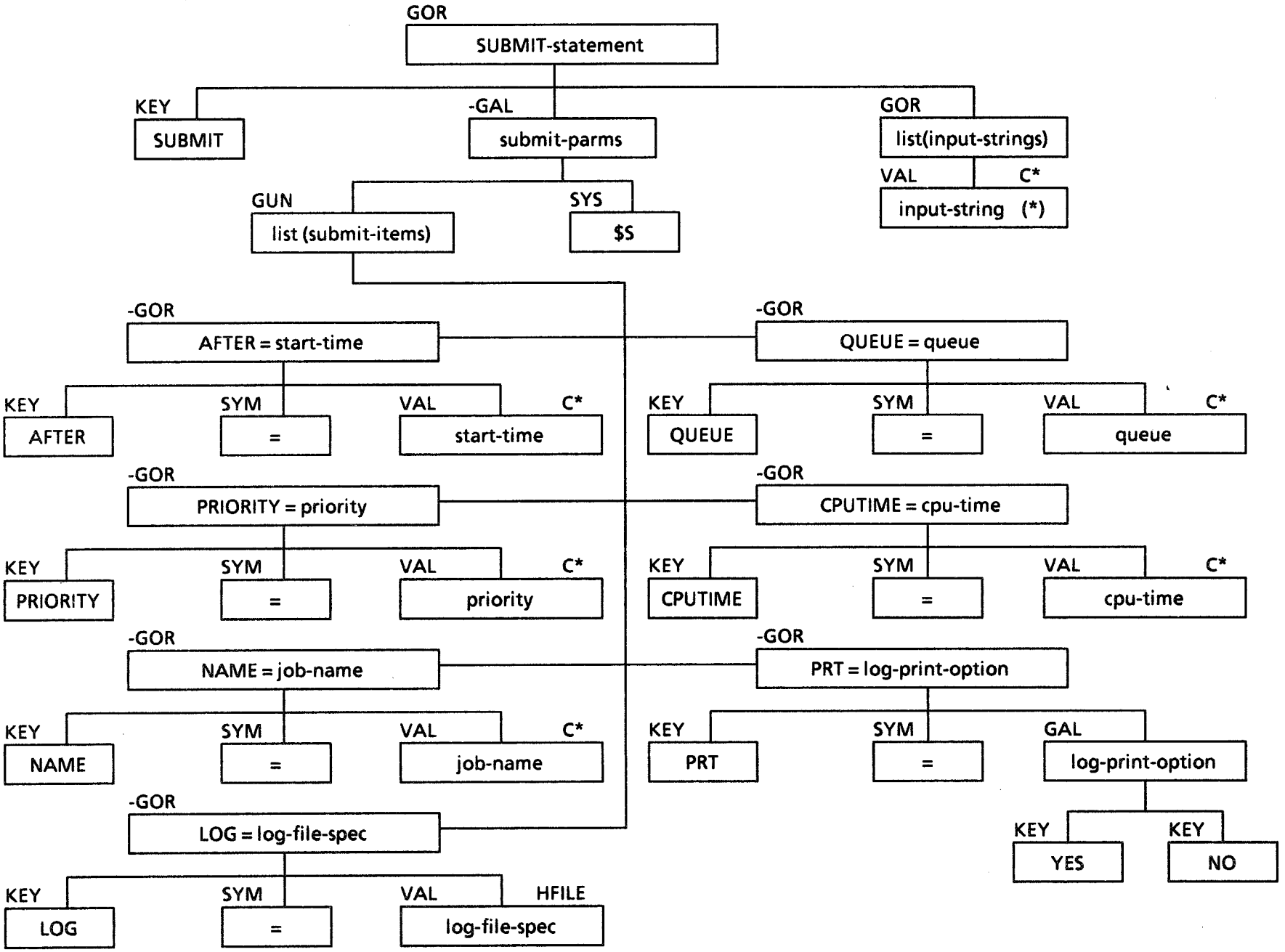
AFTER=start-time	installation defined
QUEUE=queue	installation defined
PRIORITY=priority	installation defined
CPUTIME=cpu-time	installation defined
NAME=job-name	Irunid
PRT=log-print-option	NO
LOG=log-file-spec	[default]Irunid.Log

The "Irunid" value above is a catenation of "I" with the usual unique runid generated for ACE or any other IAC module. The default log-print-option NO leaves the log-file unprinted in your default directory, and YES causes printing followed by automatic deletion. The default log-file-spec is associated with your default directory.

The "list(input-strings)" node gives the list of ACE commands to be executed in the batch run. Note that these commands are treated purely as strings, prior to initiation of the batch ACE run, and are not checked for syntax or other errors. Such checking would in fact not be possible in general, since much of the context for the batch run may not yet be determined. E.g.the existence of files, translation of macro references, etc. will in general depend upon tasks which may not yet have been performed.

```
list (input-strings)> 'RUN NAST (F=PROB3)'

list (input-strings)> ('RUN NAST (F=PROB3)'
'COST' 'HOST' '$DIR')
```

SUBMIT submit-parms list (input-strings)



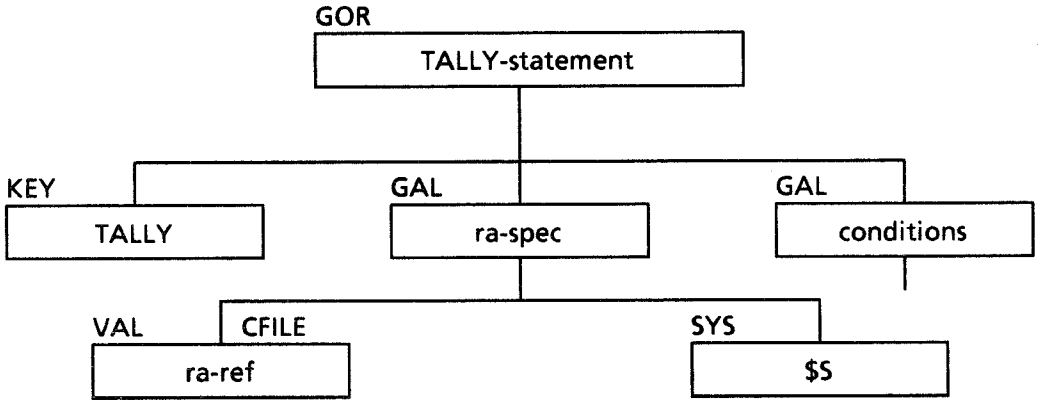
TALLY provides you with a tally of the tuples, from a relation or array, which satisfy the specified conditions.

The "ra-spec" node gives the spec of the relation or array to be tallied. A "$S" response as usual indicates that the spec is the same as the ra-spec most recently used.

You may want to use the TALLY command to determine the number of selected tuples prior to printing. You can then use the PRINT command with "$S" for the conditions node.

```
ACE>TALLY X.DAT *

ACE > TALLY X.DAT  NAME2 GT 100.

ACE > TALLY MASS.ARR +
$I1 EQN $I2 AND MASS NE 0.

ra-spec >TEMP.ARRAY

ra-spec > $S
```

TALLY ra-spec conditions



TALLY-statement

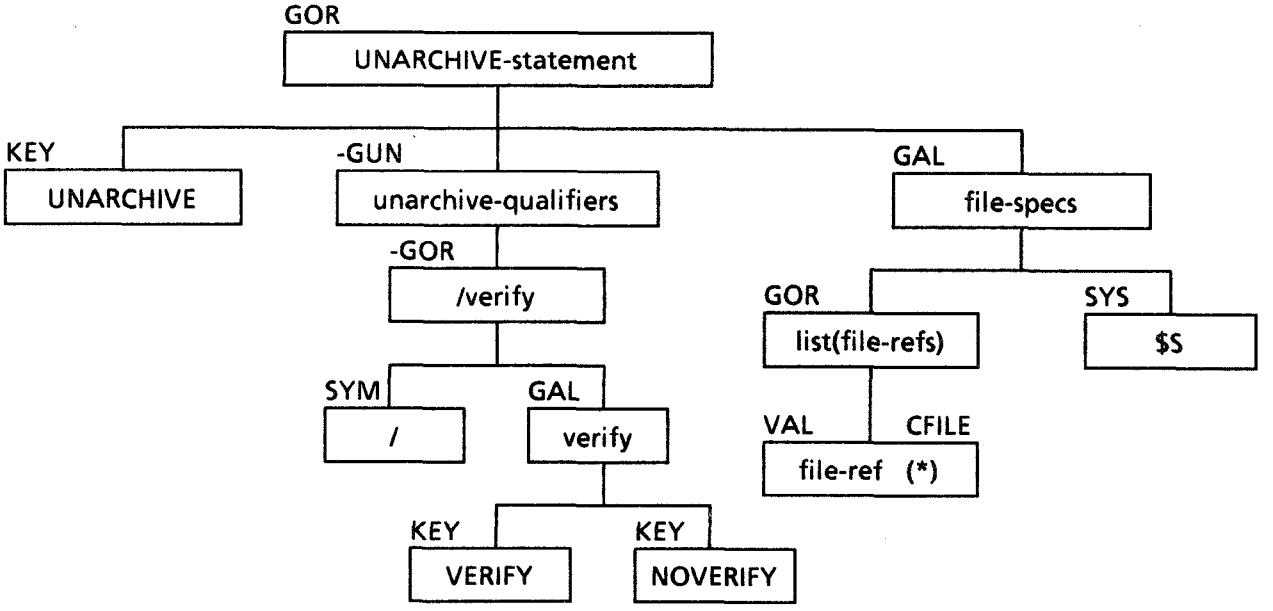
UNARCHIVE undoes the operation of the ARCHIVE command, on a list of files in the database. I.e. it sets the status flag in the IAC database catalog to N (meaning normal), and restores access to the file. The verify option (default is NOVERIFY) lets you confirm the unarchiving.

Similar to the ARCHIVE operation, it is your responsibility to restore the files to the database directory prior to using the UNARCHIVE command.

Following the UNARCHIVE command, you may want to use LIST/D to list files in the database, or GETC to obtain the database catalog for query of overall file attributes.

```
ACE>UNARCHIVE A.DAT

ACE>UNARCHIVE A.DAT, B.DAT, *.LIS

ACE>UNARCHIVE/V X.Y
```

UNARCHIVE unarchive-qualifiers file-specs

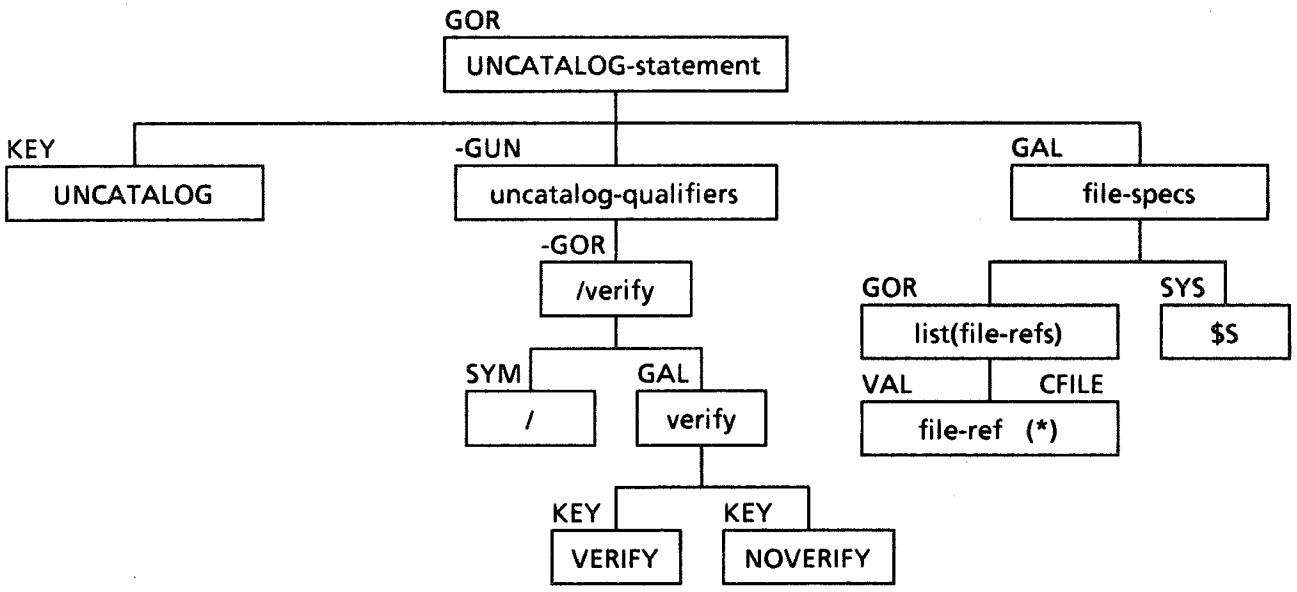


UNCATALOG removes an IAC file description from the database catalog. The associated host files (data file and/or text file) remain in the database directory after uncataloging, until and if you decide to dispose of them. The verify option (default NOVERIFY) lets you confirm results of the uncataloging.

Following the UNCATALOG command, you may want to use LIST/D to list files in the database.

```
ACE >UNCATALOG A.REL

ACE >UNCAT X:2.MAT

ACE >UNC/V X:*.DAT*
```

UNCATALOG uncatalog-qualifiers file-specs



UNCATALOG-statement

UNIQUE identifies duplicate tuples in a relation, or in one non-zero index of an array, based on comparison of specified attributes.

```
ACE >UNIQUE VARIOUS.THINGS THING1
```

The "unique-qualifiers" node provides several options. TALLY (default) or NOTALLY indicates whether the non-unique tuples are to be tallied; PRINT or NOPRINT (default) indicates whether the non-unique tuples are to be printed (if printed, they are automatically tallied); and DELETE or NODELETE (default) indicates whether the non-unique tuples are to be deleted.

```
unique-qualifiers >TALLY
unique-qualifiers >/P/D
```

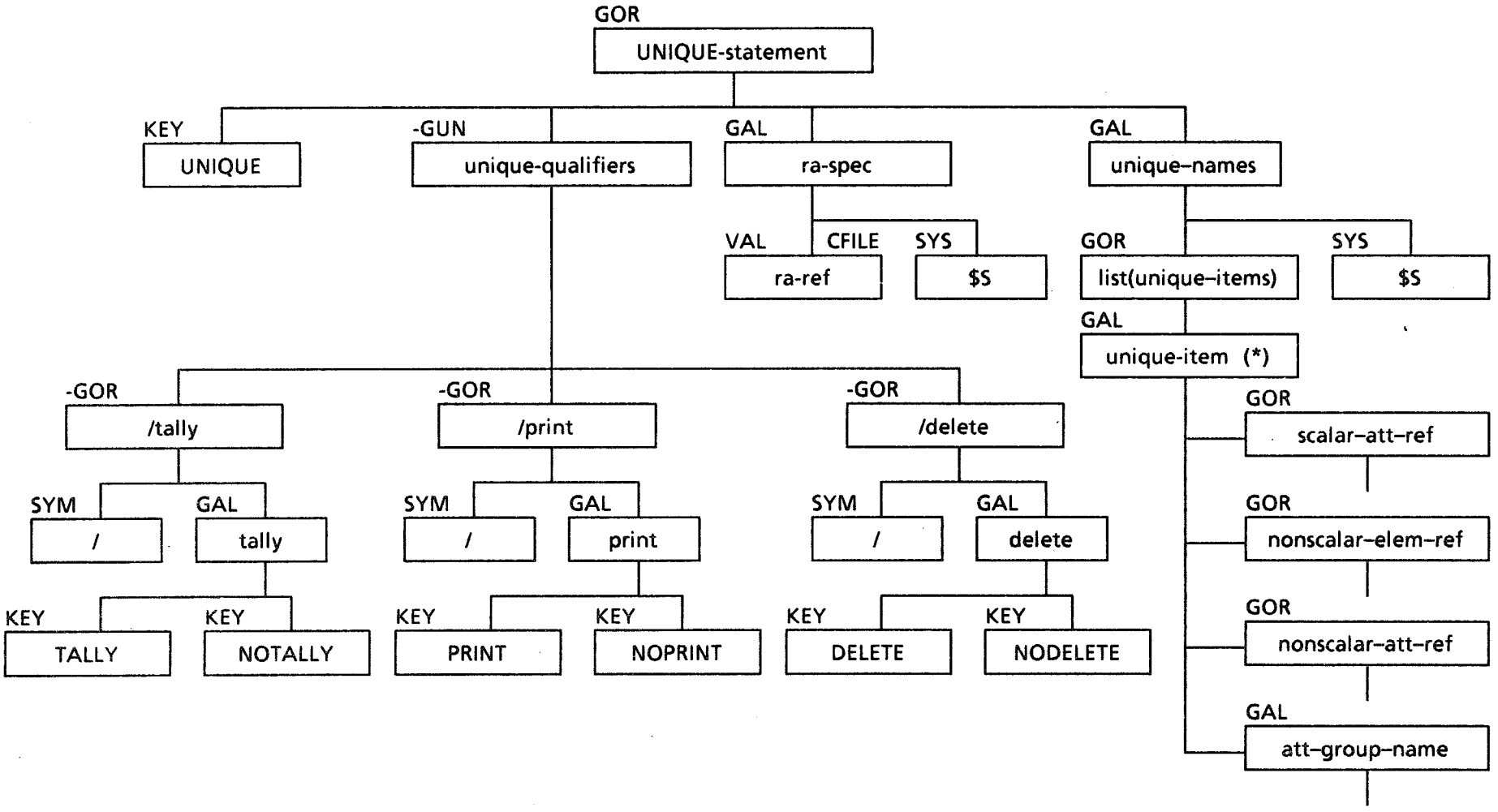
The "unique-names" node specifies the comparison attributes for determining duplicates. Operation is identical to that for "sort-names" in the SORT command, except that index names are not used. In case duplicate tuples are printed or deleted, the first occurring tuple is regarded as the "duplicated" tuple and the others as "duplicates" (the duplicates are printed or deleted).

```
unique-names >X
unique-names >X,Y,Z
unique-names >$A2
```

Each unique name may be one of several kinds of scalar or nonscalar attribute entities, as shown in the diagram.

```
unique-item >B
unique-item >A[41/80]
unique-item >AVEC[2]
unique-item >ABC[4..6]
```

UNIQUE unique-qualifiers ra-spec unique–names

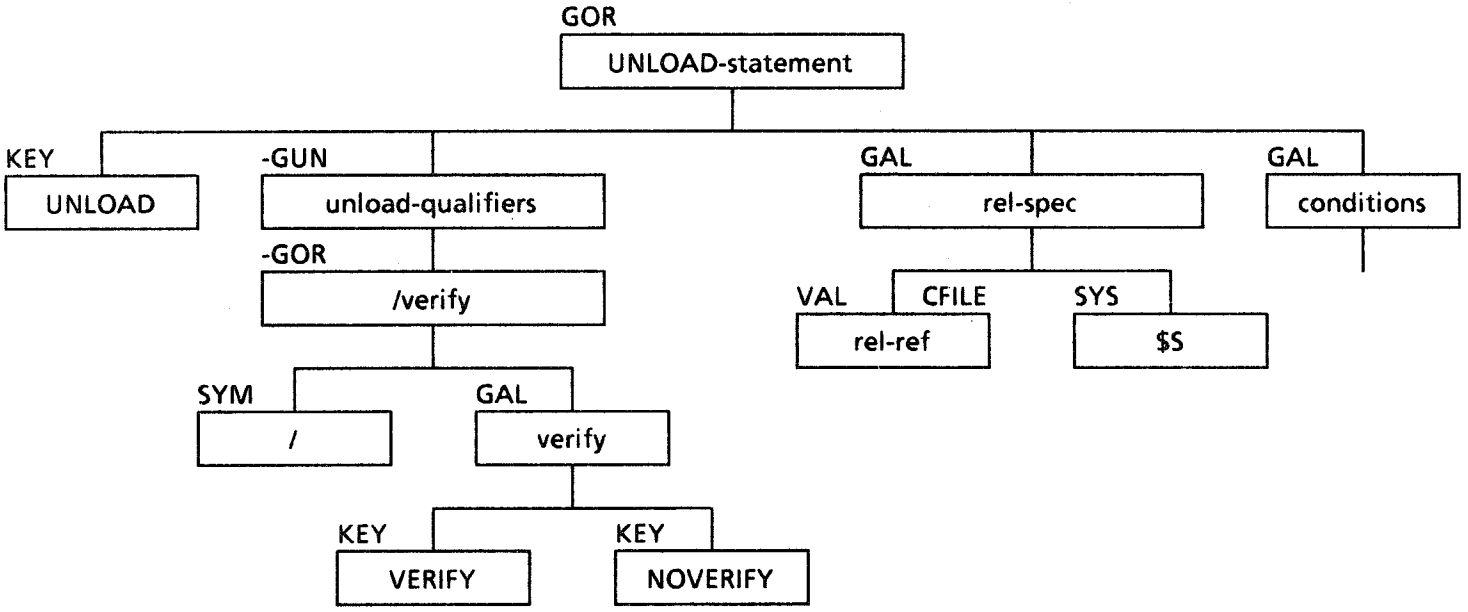


UNIQUE-statement

UNLOAD unloads (deletes) from a relation the tuples which satisfy specified conditions. The verify qualifier (default NOVERIFY) lets you confirm the unload operation by displaying the tuples which are unloaded.

```
ACE >UNLOAD A.REL  X LT 0.

ACE >UNLOAD/V A.DAT  A EQ 'XYZ'
```

UNLOAD unload-qualifiers rel-spec conditions



UNLOAD-statement

VECTORS selects vectors from an array, to be used as attribute columns in a new relation. (See also the RELATION command, for a different array-to-relation transformation capability). All vectors are logically associated with a single common index number (>0) from the array. In the examples at the right, a relation X.REL is created having the five attributes NODE,X,Y,T7 and T9. The corresponding vectors are obtained from array X.ARR, and are logically associated with the array index 1; the first three vectors are labels directly associated with index 1, and the last two vectors are the respective 7th and 9th columns of the core (index 0) array attribute TEMP. The three examples differ only in the format of the user command input; the first is totally defined by command input, and the last two illustrate the prompted input alternative which you may use instead of the "vectors" node.

```
(ACE >VECTORS X.ARR 1 +
(NODE,X,Y,TEMP 7=T7,TEMP 9=T9) X.REL

ACE >VECT X.ARR 1 X.REL
vector 1 >NODE
vector 2 >X
vector 3 >Y
vector 4 >TEMP 7=T7
vector 5 >TEMP 9=T9
vector 6 >$

ACE >VECT X.ARR 1 X.REL
vector 1 >NODE,X,Y
vector 4 >TEMP 7=T7,TEMP 9=T9 $
```

You may use the optional "vectors" node to select vectors from the array, and corresponding attribute names for the relation, as in the first example above. If you don't provide this node, you will interactively prompted for the vectors, as in the next two examples. Note that the "$" terminates interactive selection of vectors. In either case, you may select a variety of different index and attribute related vectors to be included in the new relation. At least until you become experienced in using the VECTORS command, you may find the prompted input alternative is simpler and less error prone.

```
vectors >(NODE,X,Y,TEMP 7=T7,TEMP 9=T9

vector 1 >NODE,X,Y
vector 4 >TEMP 7=T7,TEMP 9=T9 $

vectors >$I1=I,A 3=X,A 4=Y

vector-item >XLABEL[..]
vector-item >$IMAG(ZVAL) 6=ZR6/'E12.4'
```

The "arr-index-id" node specifies the array index with which the selected vectors are to be associated, and indicates the direction in which the vectors are taken.

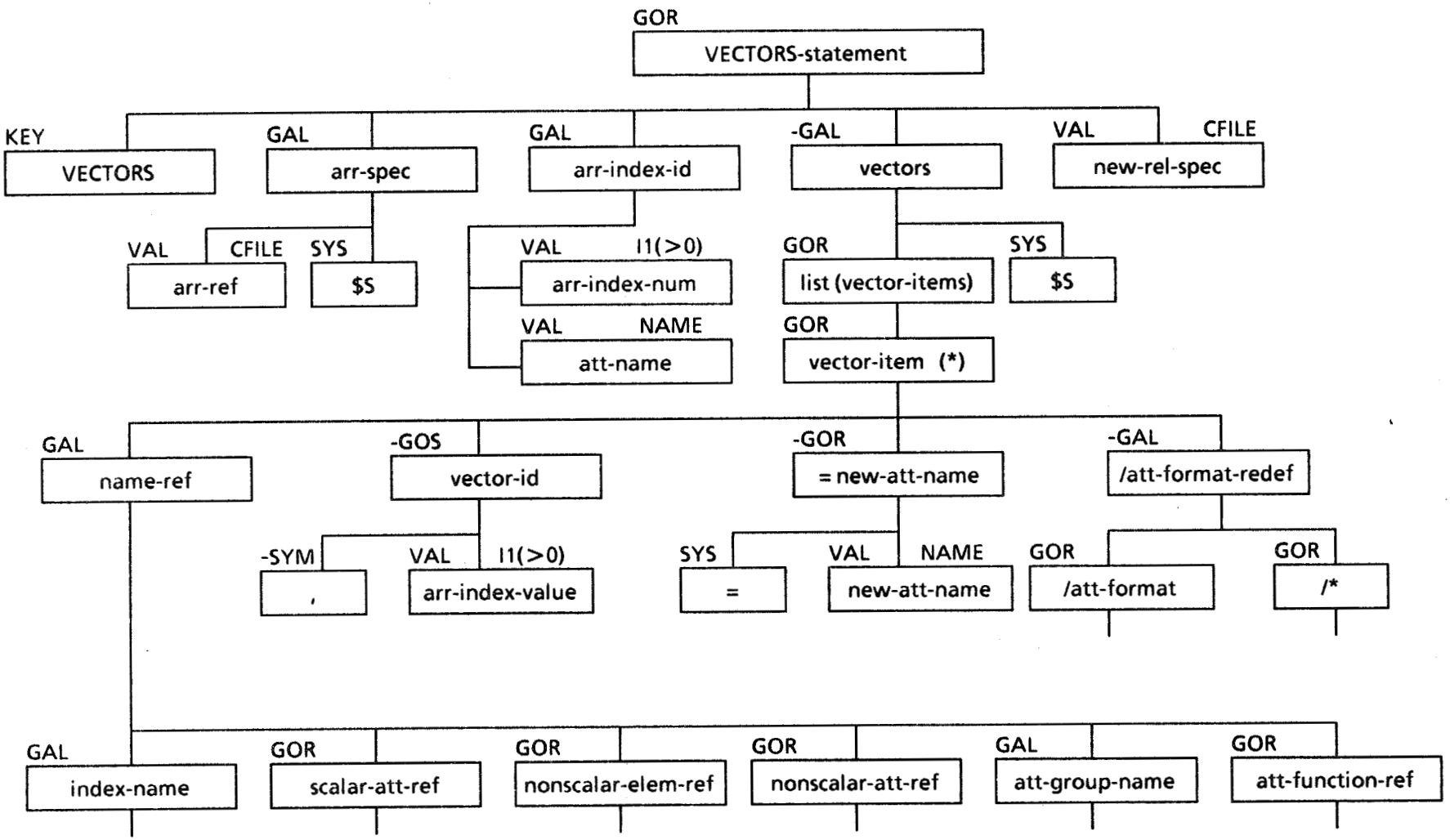
```
arr-index-id >2

arr-index-id >TIME
```

The "vector-id" node selects a vector from within a core (index 0) array attribute. This node consists of a number of integer index values, the number being one less than the array dimension.

```
vector-id >17

vector-id >11,23
```

Note - The prompted input alternative to the "vectors" node is not currently operational.

VECTORS arr-spec arr-index-id vectors new-rel-spec

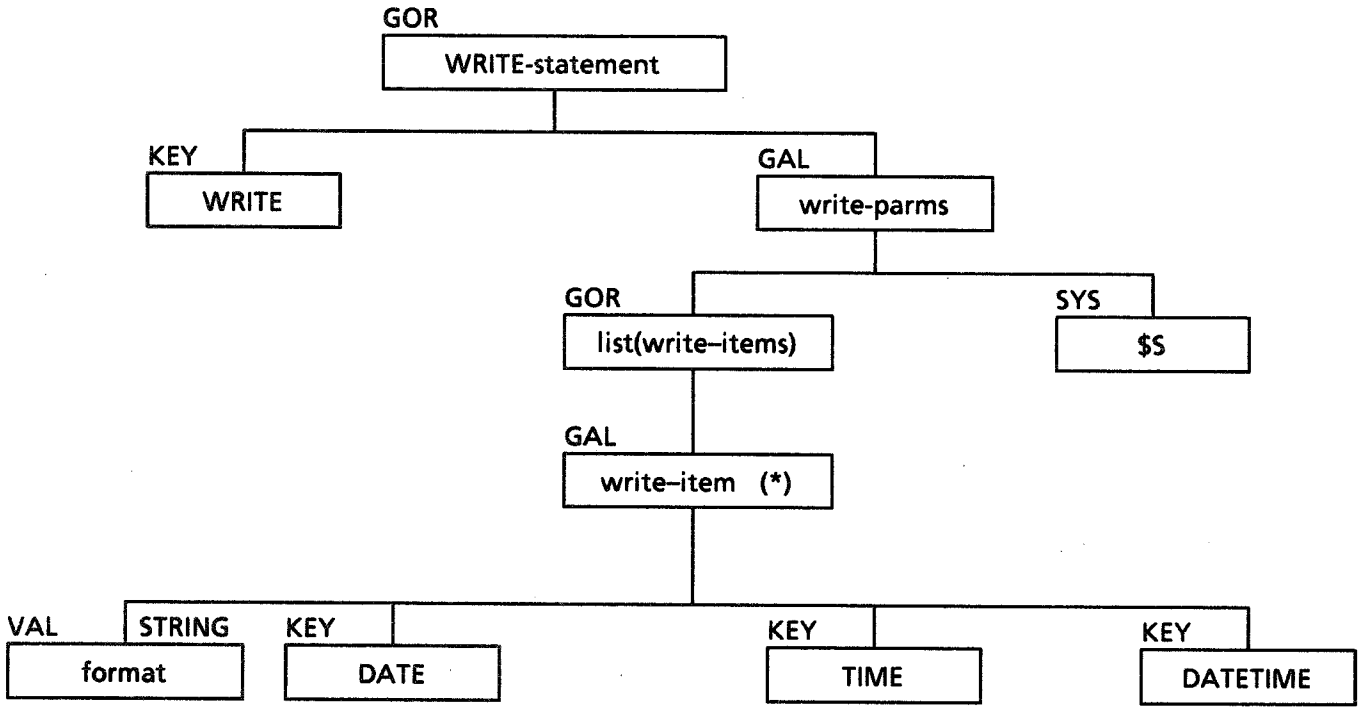


WRITE allows you to write one or more items onto the currently active OUTPUT file. The "format" node causes an empty FORTRAN write list to be executed, using the specified format.

Note - The DATE, TIME and DATETIME items are not currently implemented.

```
ACE >WRITE '''This is my note'''
ACE >WRITE '2H$$, ''***HEADING'''
ACE >WRITE ('''LINE 1''', '''LINE 2''')
ACE >WRITE $S
```

WRITE write-parms



WRITE-statement

2.2 MACROS AND AMACROS

Two ways are provided by ACE to expand the user's command. Macros replace specially identified 'macro calls' with text defined previously. In this way, long and frequently used strings of text can be entered with a shorter macro call, or a definition be set to be used by macro calls in a command file. On the other hand, amacros replace their 'amacro calls' with the input supplied by the user in response to a prompt. This has obvious uses in a standard command file which could be applied to any of a number of datasets: the user could fill in which is needed during execution without editing the command file.

Macro and amacro calls are identified by the use of special characters to delimit the reference. Replacement occurs sequentially, left to right, in the command line. Both macros and amacros may contain other macro and amacro calls within the replacement text. These nested calls are evaluated when the text containing them has replaced the original reference.

The following topics deal with the syntax of macros and amacros, and with a more thorough description of macro usage. Please note that because of this syntax, use of the characters "@" and "?" at any time within ACE must be done with the consideration of these rules. Even the appearance of these two characters within a quoted string will not avoid their possible interpretation as part of a macro or amacro reference.

Macro Syntax and Overview - ACE will interpret all characters between two single @ characters as a macro call. However, two adjacent @'s are condensed into a single @, which is then treated as text and not as one of the call's delimiters. (For example, "@@COMFILE" could be used for the invocation of a command file on the host system.) In its simplest form, a macro call consists of:

@ID-value@

A relation called MACRO.RELATION must have been previously defined with two attributes named ID and TEXT, both specified as Cn or C* type. ACE searches this table for a match between the ID-value supplied by the user and a value

in the ID attribute; if successful, the string @ID-value@ is replaced with the string value of TEXT in the corresponding position in the table. No blanks are inserted that are not part of the TEXT string value.

The more complete form of the macro call, explained in the final topic of this section, is:

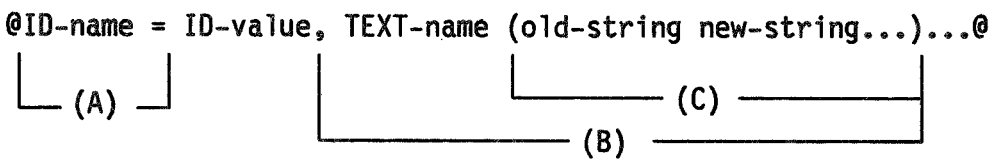
@ID-name = ID-value, TEXT-name (old-string new-string...)...@

Amacro Syntax and Overview - ACE interprets all characters between two single ? characters as an amacro call. Two adjacent ?'s are condensed into a single ? which is then treated as text, not one of a call's delimiters. An amacro call consists of:

?prompt-string?

The entire string, including the delimiting question marks and any condensed marks, appears as a user prompt, followed by a prompting carat. The user must then enter the appropriate text, which then replaces the amacro call string in the command line. If the input text is to be treated as one character literal but contains blanks or special characters then it should be enclosed in single quotes.

Detailed Consideration of Macro Usage - The simple form of the macro given previously is most useful for abbreviations. Much more powerful capability is offered in the complete syntax, described below. The related SET MACRO command allows other tables besides MACRO.RELATION to be used to interpret the macro calls, and these tables may have many attributes. The SET MACRO command also allows the default attribute names to be changed from ID and TEXT, and/or allows the macro recognition and interpretation process to be turned off.



Only the ID-value is required within the @ characters. The following options may be used to fill in the other items.

(A) This option gives the name of the attribute to be searched for an exact match with ID-value. The equal sign is required. The name defaults to ID.

(B) A replacement string is generated for each occurrence of this option, using the value of the named text attribute. If the option is omitted, one generation from the attribute named TEXT is used. The comma is required for each description of a generated string. The TEXT-name is optional if substitution pairs (option C) are present, defaulting to TEXT.

(C) Portions of the text attribute value can be altered by the macro call supplied by the user. Within the required parentheses are pairs of substitution strings. The text attribute value is scanned from left to right for a substring matching any of the old-string values given; every successful match is replaced by the corresponding new-string (but the scan then continues just beyond the new-string, to look for further old-string matches). In cases of ambiguity such as P1 and P11, the longer match is chosen.

Note that amacros embedded within macros, either in the call or in the text value, are not evaluated until after the macro call is evaluated and replaced in the command line. Therefore a call such as "@VALUE,?Which text?@" will not work; the amacro will not be resolved before the macro looks for the text attribute name. The amacro could be placed in a substitution option, but each substitution will cause the prompt - one prompt does not fill in all the replacements. A way around this problem is given in Example 2.

Examples - The following four examples illustrate the usage of ACE macros and amacros.

Example 1. Interactive use of standard files makes abbreviation for file names convenient. The user might enter these ACE commands:

```
DEFINE RELATION MACRO.RELATION (ID,C*/A5 TEXT,C*/A30)
LOAD MACRO.RELATION N,'SATELLITE.NASTRAN'$
GET @N@
LOAD @N@, ...
PUT @N@
```

Example 2. A command file has been set up to refer to a file via one amacro call to name it and several subsequent macro calls. Assuming the relation for the macros has been defined as in Example 1, the command file might contain:

```
UNLOAD MACRO.RELATION ID EQ #FILE#
LOAD MACRO.RELATION #FILE#,?WHICH FILE? $
GET @#FILE#@
SHOW FILE (@#FILE#@)
PUT @#FILE#@
SET IO INPUT = '*'
```

Note the amacro in line 2 which will prompt the user who invokes the command procedure.

Example 3. Several files with similar names are frequently generated. Rather than have an abbreviation for each in the macro relation, substitution within the macro call is used. Assuming the relation has been loaded with "STD,'BASECASE.STD'" the call might be:

```
GET @STD, (D D1)@      becoming GET BASECASE.STD1
```

Example 4. This is a contrived example to display the power of macros.

```
DEFINE RELATION MACRO.MINE (ID,C*/A5 TEXT,C*/A30 ALT,C*)
SET MACRO (REL=MACRO.MINE)
LOAD MACRO.MINE A,'MACRO.RELATION','MACRO.MINE',
                B,'@@A,(''.'' '':2.'')@@','??NUMBER??'$
PRINT @A@ $A *
PRINT @A,(N 'N $A *')@     resulting in the same as the prior line
PRINT @A,ALT@ $A *
PRINT @B,(2 '@@@@B,ALT@@@@')@     which causes the prompt ?NUMBER? to
appear and the reply to be included in PRINT MACRO:reply.RELATION.
```

3.0 MODULES

Modules available within IAC are of four general types:

1. Major technical modules (e.g. MSC NASTRAN and DISCOS) which provide standalone computational capability as well as support for various multi-discipline solution paths;

2. Interface modules (e.g. INDA and INSA) which provide required data flow in the solution paths;

3. Special purpose modules such as ACE and PLOT; and

4. User provided modules.

You will usually execute an IAC module via an executive RUN command, which defines appropriate parameter specifications. You may, however, accomplish the execution in a standalone mode, via direct definition of the module associated parameter file. This section defines the requirements and options for RUN command execution of each of the currently available IAC modules. Section 7 (Module Implementation) describes the approach for implementing modules within IAC.

3.1 MSC NASTRAN/TECKPLOT

MSC NASTRAN (Reference 7) is a general purpose finite-element analysis program with emphasis in the structural and thermal technologies. This section describes the IAC implementation of MSC NASTRAN, and the associated graphics module TECKPLOT. The general RUN schematic for executing these modules within ACE is shown in Figure 3.1-1. The listed RUN statement parameters and diagrammed files describe all of the specific IAC-provided NASTRAN capabilities. The following is a summary of NASTRAN RUN parameters which may be specified.

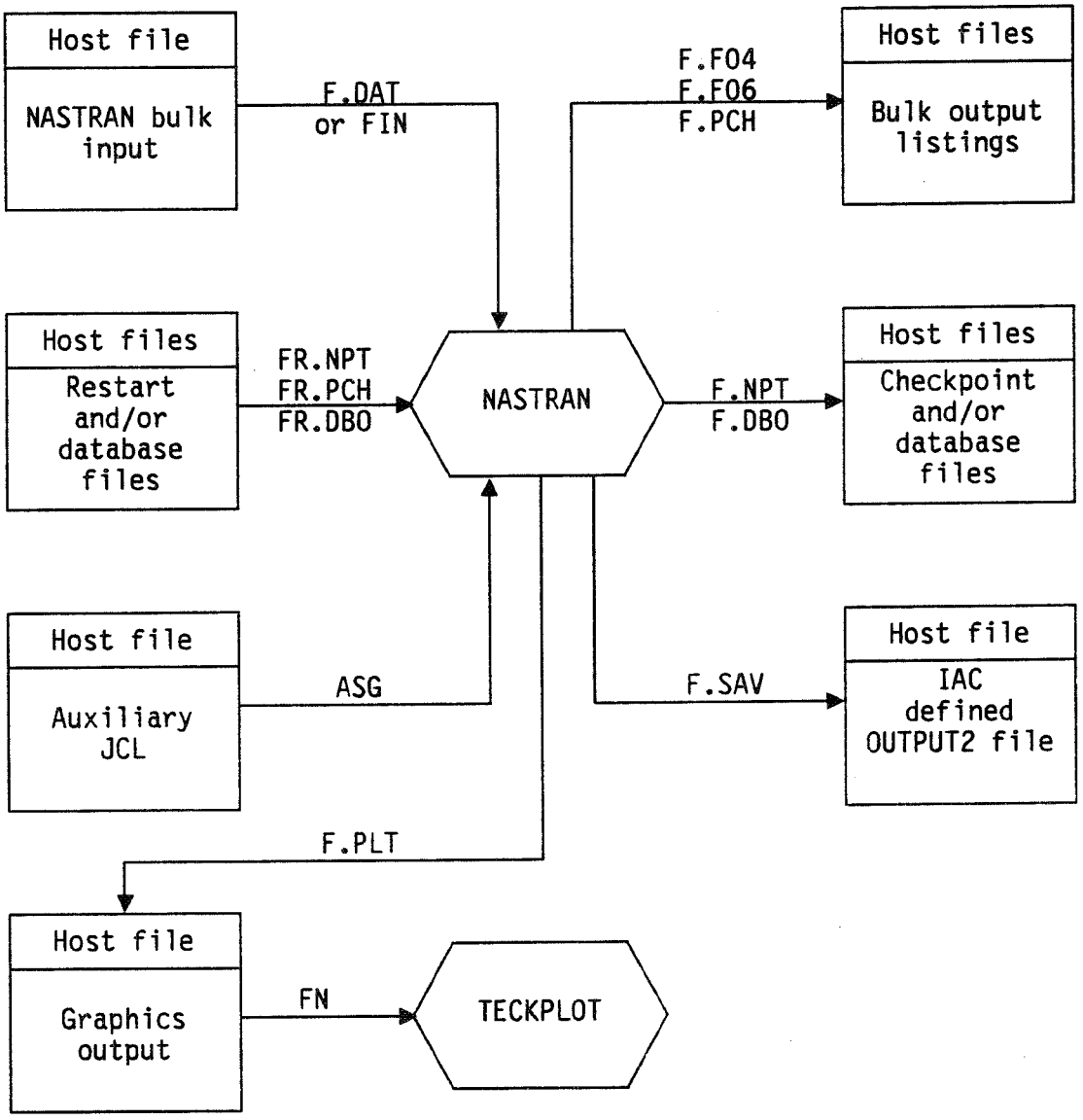
F	=	basic file name	
FIN	=	input file spec	
PRT	=	print option	(default = NO)
RFA	=	Rigid Format Alter id's	
FR	=	restart file name	
ASG	=	auxiliary JCL file spec	
WSL	=	working set size	(default = installation defined)
SCRATCH	=	scratch directory spec	(default = installation defined)
DBDISP	=	database disposition option	(default = KEEP)

However, it can be expected that many of these capabilities will be used only infrequently. A frequently used abbreviated form of the RUN statement is

```
RUN NASTRAN (F = name, PRT = option)
```

Any MSC-supplied Rigid Format sequence may be executed from within IAC, using this command. The Rigid Format number is specified on the SOLUTION card in the executive control portion of the input file.

The F parameter in the RUN statement is used to name basic NASTRAN files. As shown in Figure 3.1-1, file type DAT is the bulk input file; F04, F06 and PCH are output files which respectively contain message (log) information, bulk output listing and punch output (punch output may include checkpoint dictionary); SAV is a NASTRAN OUTPUT2 file used to store output from an IAC




```
RUN NASTRAN (F=name, FIN=spec, PRT=option, RFA=(id-list)+
            ,FR=name, ASG=spec, WSL=size, SCRATCH=spec+
            ,DBDISP=option)

RUN TECKPLOT (FN=spec)
```

Figure 3.1-1: MSC NASTRAN/TECKPLOT RUN Schematic

Rigid Format Alter sequence (see RFA parameter); and PLT contains output plot data.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

As an example, the user might input the statement

```
RUN NASTRAN (F = PLATFORM, PRT = YES)
```

and provide an input file which contains a "SOLUTION 24" card. This would cause a NASTRAN structural statics analysis to be performed. Input would be read from the file PLATFORM.DAT. The output would be temporarily written on files (bulk output on PLATFORM.F06 and solution log on PLATFORM.F04), until these files are printed on the line printer. (File type specifications such as DAT, F04 and F06 are MSC naming conventions for running NASTRAN on the VAX computer. NASTRAN permanent output files are normally generated in the user's host directory.)

The complete set of NASTRAN parameters is given by the RUN statement shown in Figure 3.1-1.

FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.

The RFA parameter allows any Rigid Format Alter sequence (special IAC-defined, standard MSC-supplied or arbitrary user-defined) to be executed, and may be multiple valued.

IAC currently provides several special Alter sequences, in order to accomplish specific data flow requirements. The RFA id's and associated definitions are as follows.

RFA = IACDMAP1 (normal modes, Section 3.1.2)
RFA = IACDMAP2 (thermal nonlinear steady state, Section 3.1.1)
RFA = IACDMAP3 (thermal transient, Section 3.1.1)
RFA = IACDMAP5 (static deformation, Section 3.1.2)

For a standard MSC-supplied Alter, the Rigid Format number is given on the input file SOLUTION card, and the RFA parameter specifies the Alter name (consistent with the usual MSC naming convention, the RFA id is RFxxDxx, where x is a digit).

The IAC NASTRAN JCL first attempts to identify an RFA id as one of the special IAC names, e.g. IACDMAP1. If not successful, it then attempts to identify the id as one of the standard NASTRAN Alter names. If still not successful, it attempts to use the id value as the complete spec of a file which contains a user defined Alter sequence.

The FR parameter gives the name (or optionally the entire leading portion of a file spec, down through the name) for restarting from an existing NASTRAN database file (type DBO), or from an existing checkpoint file (type NPT) and associated punch file (type PCH).

ASG gives the spec of a command file containing an auxiliary set of JCL, which is to be executed during the course of the NASTRAN run. The auxiliary JCL is executed just after the standard IAC NASTRAN JCL file assignments.

WSL defines the working set size (number of pages of actual memory) for the run.

SCRATCH allows the user to specify a host directory for storing scratch files during NASTRAN execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

DBDISP provides an option for disposing of a NASTRAN created database; the keyvalue KEEP (default) keeps the database in existence, and DELETE deletes it.

The MSC implementation of NASTRAN on the VAX was designed for a batch mode of execution. Within this implementation, all NASTRAN jobs were intended to be submitted to a single queue, so that only one NASTRAN job is in execution at a time. (This was a practical restriction based on the host resources used by NASTRAN). In the IAC implementation of NASTRAN, the NASTRAN JCL has been modified to permit interactive execution via the IAC RUN command, i.e. execution where the user remains in the loop with a certain amount of control, and is informed when the NASTRAN execution has been completed. Also, it is possible to submit a batch job via the IAC SUBMIT command to one of several different queues. However, it is recommended that the IAC user run long NASTRAN jobs in the batch mode, submitting them to a special installation-provided NASTRAN queue.

In order to implement IAC required solution paths, some specific NASTRAN thermal and structural computational sequences have been developed. The following two sections give details for using these sequences.

3.1.1 NASTRAN THERMAL

IAC provides two special Rigid Format Alter sequences, denoted by IACDMAP2 and IACDMAP3, respectively, for nonlinear steady-state and transient thermal analysis. They can be used either for standalone analysis, or as part of the IAC Solution Path II or IV (see Section 4). These sequences provide for various time-dependent properties and computed data to be written to a NASTRAN OUTPUT2 file name.SAV. Time values are output only for the transient run, and correspond to the time interval specified on the NASTRAN input TSTEP card. The interface module INTA (see Section 3.7) can later be used to read this file and insert appropriate data into the database. In executing a NASTRAN thermal analysis with these Rigid Format Alter sequences, only one input load case is handled in a single thermal analysis run.

Steady-State - The steady-state run is executed using a statement of the form

```
RUN NASTRAN (RFA = IACDMAP2, F = name)
```

The NASTRAN input executive control must contain a "SOL 74" card to identify the run as nonlinear steady-state. It must also contain a "READ 9" card to access the IACDMAP2 Rigid Format Alter sequence; this card should appear prior to the CEND card, and prior to any user defined Alters.

Transient - The transient run is executed using a statement of the form

RUN NASTRAN (RFA = IACDMAP3, F = name)

The NASTRAN input executive control must contain a "SOL 89" card to identify the run as transient. It must also contain a "READ 9" card to access the IACDMAP3 Rigid Format Alter sequence; this card should appear prior to the CEND card, and prior to any user defined Alters.

3.1.2 NASTRAN STRUCTURAL

Normal-Modes - In addition to the usual standalone NASTRAN structural capabilities, IAC provides a special DMAP sequence, denoted by IACDMAP1, for normal-modes analysis. It is generally used as part of the IAC Solution Path III or IV (see Section 4). This sequence provides for dynamic normal-modes data to be written to a NASTRAN OUTPUT2 file name.SAV. The interface module INDA (see Section 3.8) can later be used to read this file and insert appropriate data into the database. The sequence is executed using a statement of the form

RUN NASTRAN (RFA = IACDMAP1, F = name)

The NASTRAN input executive control must not contain a "SOL" card. However it must contain a "READ 9" card to access the special DMAP sequence; this card should appear prior to the CEND card.

Static Deformation - IAC provides a special Rigid Format Alter sequence, denoted by IACDMAP5, for static deformation analysis. It can be used either for standalone analysis, or as part of the IAC Solution Path II or IV (see Section 4). This sequence provides for nodal and/or modal displacements, and various other element/nodal data, to be written to a NASTRAN OUTPUT2 file

name.SAV. The interface module INSA (see Section 3.11) can later be used to read this file and insert appropriate data into the database. The run is executed using a statement of the form

```
RUN NASTRAN (RFA = IACDMAP5, F = name)
```

The NASTRAN input executive control must contain a "SOL 24" card to identify the run as statics. It must also contain a "READ 9" card to access the Rigid Format Alter sequence; this card should appear prior to the CEND card, and prior to any user defined Alters. Data written to the OUTPUT2 file depends on particular case-control, load sets, scalar-point freedoms, and multi-point constraints defined in the NASTRAN bulk input file. Processing of the OUTPUT2 file by INSA is controlled by two NASTRAN DTI type tables, named IACTIME and IACMODE, defined in the NASTRAN bulk input file. NASTRAN does not use these tables directly, but simply passes them on to the OUTPUT2 file. IACTIME correlates time values with solution subcase id's, and IACMODE correlates mode id's with scalar-point id's. The NASTRAN IACDMAP5 sequence and associated INSA interface module provide the following options, listed according to the tables (IACTIME and/or IACMODE) which the option requires in the bulk input.

1. IACTIME - The interface module INSAT (see Section 3.9) will usually be used to read a node/time/temperature array from the IAC database, and automatically generate NASTRAN statics input data consisting of the IACTIME table and the thermal load cases. NASTRAN IACDMAP5 computes deformations for specified thermal (and/or mechanical) loadings, via a series of one or more solution subcases. Data written to the OUTPUT2 file allows INSA to create in the database various nodal and element associated data.

2. IACMODE - The interface module INSAM (see Section 3.10) will usually be used to read a mode-shape definition array from the IAC database, and automatically generate NASTRAN statics input data consisting of the IACMODE table, scalar-point freedoms (one for each mode) and multipoint constraints (one for each nodal freedom, making that freedom dependent upon the modal freedoms). NASTRAN IACDMAP5 uses

the MPC's to perform a nodal-to-modal basis conversion (similar to static condensation) on the input model, so that deformations are restricted to a combination of the mode shapes. Data written to the OUTPUT2 file allows INSA to create in the database various nodal, modal and element associated data.

3. IACTIME and IACMODE - When both tables are present in the NASTRAN bulk input, the combined capabilities of options 1 and 2 are available. Data written by NASTRAN IACDMAP5 to the OUTPUT2 file allows INSA to create in the database various nodal, modal and element associated data.

4. When neither the IACTIME nor IACMODE table is present in the NASTRAN bulk input, NASTRAN IACDMAP5 computes nodal deformations as in option 1. Data written to the OUTPUT2 file allows INSA to create in the database various nodal and element associated data.

3.1.3 TECKPLOT EXECUTION

TECKPLOT provides interactive display of Tektronix-like plots from a NASTRAN generated plot file. Additional information is provided in the MSC NASTRAN Application Manual (Reference 2). The following TECKPLOT RUN parameter is required.

FN = plot file spec

The FN parameter defines a complete file spec for the input plot file.

3.2 DISCOS

DISCOS (Dynamic Interaction Simulation of Controls and Structure) is a very general system dynamics module for time-domain and frequency-domain analysis. It analyzes multiple-flexible-body spacecraft, including nonlinear gyrodynamics and controls effects, but subject to linear stiffness definition for each body. Appendix C describes IAC-associated DISCOS enhancements, including the nametag/id free-field input option, the generation of the A,B,C system definition matrices, and the time-domain interactive graphics. Major aspects of the DISCOS theory and computational flow are provided in Reference 8. The general run schematic for executing DISCOS within ACE is shown in Figure 3.2-1. The following is a summary of DISCOS RUN parameters which may be specified.

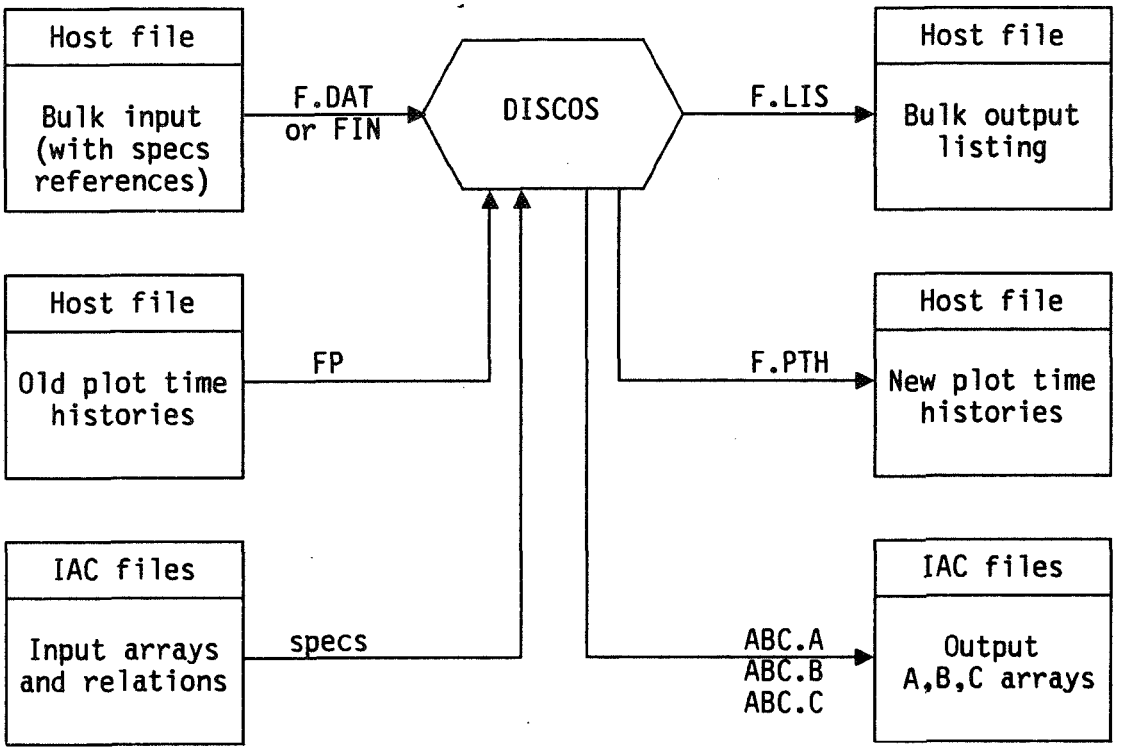
```
F       = basic file name
FIN     = input file spec
FP      = plot file spec
EXE     = user load module spec
ITAG    = input file layout          (default = installation defined)
AREA    = data storage area          (default = DH)
ABC     = ABC matrix name
PRT     = print option               (default = NO)
SCRATCH = scratch directory spec     (default = installation defined)
```

A typical abbreviated form of the RUN statement is

```
RUN DISCOS (EXE=spec, F=name)
```

The F parameter names the input file (name.DAT), the output file (name.LIS), and the plot-time-history file (name.PLT).

FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.



RUN DISCOS (F=name, FIN=spec, FP=spec, EXE=spec, ITAG=option+
,AREA=code, ABC=name, PRT=option, SCRATCH=spec)

Figure 3.2-1: DISCOS RUN Schematic

FP gives the spec of a previously created plot-time-history file to be used as input for the current run in order to perform additional plotting. A new plot file is created if and only if the FP parameter is not given.

EXE is an optional parameter; if given it defines a complete spec for a user-defined load module, to be executed instead of the standard DISCOS module.

ITAG has the value TAG or NOTAG (default = installation defined) and defines the layout of the bulk input file. TAG indicates nametag/id layout, and NOTAG indicates no-nametag layout. For the two respective input file descriptions, see Appendix C of the IAC user manual, and the source listing of the DISCOS DEF3 routine.

The AREA parameter gives the storage area from/to which IAC files are to be transferred. D indicates database; H indicates host directory; DH (default) indicates database if possible, else host directory; and HD indicates host directory if possible, else database. For example, a DISCOS array read operation following the default AREA=DH specification results in an attempt to retrieve the array from the IAC database; if retrieval is not successful (e.g. database not open, invalid IAC file spec, or file not found), an attempt is then made to retrive the array from the host directory.

ABC is an optional parameter which gives the name of the output plant definition matrices (ABC.A, ABC.B, and ABC.C). These matrices are generated if and only if ABC is specified. They are stored in the host directory or the IAC database, depending upon the value of the AREA parameter. See Appendix C for details of the associated system model linearization and the A,B,C matrix generation prcedures.

The PRT parameter controls printing of the output listing file and equals one of the key values YES, NO (default), or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue with later printing by the user followed by automatic deletion from the user's directory.

SCRATCH allows the user to specify a host directory for storing scratch files during DISCOS execution, with the default being an installation dependent scratch directory; the scratch files are automatically deleted after execution is completed.

As an example, the user might input the ACE command

```
RUN DISCOS (EXE=DISCX.EXE, F=DISH)
```

This would cause the user's DISCOS executable DISCX.EXE to be run, using the input file DISH.DAT, and forming the output file DISH.LIS and (assuming that a non-zero plot interval was specified in the DISCOS input data) the output plot-time-history file DISH.PTH.

At a later time, the user might input the command

```
RUN DISCOS (EXE=DISCX.EXE, F=DISH, FP=DISH.PTH).
```

This would cause DISCOS to be run in the interactive plotting mode, using the previously generated plot-time-history file DISH.PTH, without any further analysis.

3.3 TRASYS/TRASPLOT

This section describes the IAC implementation of the TRASYS II (Thermal Radiation Analysis SYStem) module, and the associated graphics module TRASPLOT (References 9-11). The general RUN schematic for executing these modules within ACE is shown in Figure 3.3-1. The listed RUN statement parameters and diagrammed files describe all of the specific IAC-provided TRASYS capabilities. The following is a summary of the TRASYS RUN parameters which may be specified.

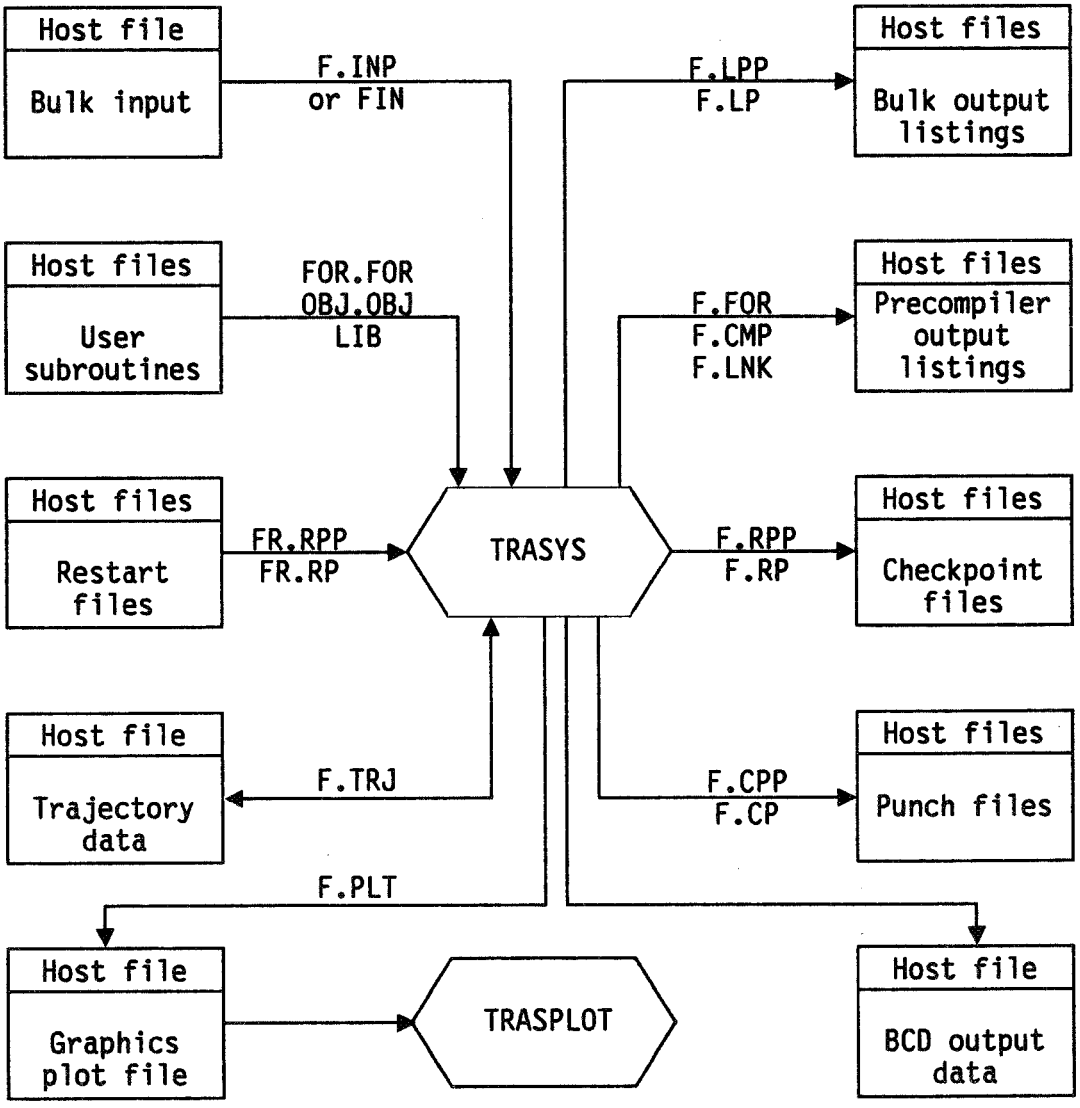
F	=	basic file name	
FIN	=	input file spec	
PRT	=	print option	(default = NO)
FR	=	restart file name	
FOR	=	user source names	
OBJ	=	user object names	
LIB	=	user library specs	
LIST	=	listing generation options	(default = NOFOR,NOCMP,NOLNK)
SCRATCH	=	scratch directory spec	(default = installation defined)

A frequently used form of the TRASYS RUN statement is

```
RUN TRASYS (F = name, PRT = option, FR = name)
```

This statement executes a two-part TRASYS module, consisting of the preprocessor to generate an executable processor program, and the processor to perform the analysis.

The F parameter in the RUN statement is used to name basic TRASYS files. As shown in Figure 3.3-1, file type INP is the bulk input file; LPP and LP are output listing files from the preprocessor and processor, respectively; the TRJ file is used to handle trajectory data; the FOR, CMP and LNK files are output FORTRAN, compiler and linker listings, respectively, resulting from the preprocessor execution; RPP and RP are checkpoint files which can be used to save results from the preprocessor and processor, respectively; BCD




```
RUN TRASYS (F=name, FIN=spec, PRT=option, FR=name+
           ,FOR=(name-list), OBJ=(name-list), LIB=(spec-list)+
           ,LIST=(option-list), SCRATCH=spec)

RUN TRASPLOT
```

Figure 3.3-1: TRASYS/TRASPLOT RUN Schematic

contains BCD output data; CPP and CP are, respectively, preprocessor and processor output punch files, which may contain data for later input to a thermal analyzer such as SINDA or NASTRAN; and PLT is the plot data file for input to TRASPLOT.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

FR gives the name (or optionally the entire leading portion of a file spec, down through the name) for restarting from existing checkpoint files (types RPP and RP).

The complete set of TRASYS parameters is given by the RUN statement shown in Figure 3.3-1.

FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.

FOR, OBJ and LIB identify user subroutines in the form of source code, object code and libraries, respectively. These subroutines are linked into the TRASYS module prior to execution; in case of duplicate subroutine names, user routines will replace the corresponding TRASYS routines. Note that for user libraries, LIB defines a complete link type of spec which may include the directory and type and must include "/LIB".

LIST controls generation of the preprocessor output listings (printing is controlled by PRT); it is a keyvalue list which may contain FOR or NOFOR, CMP or NOCMP, and LNK or NOLNK, to specify generation or no generation of the respective FORTRAN, compiler and linker output files (compiler and linker files will be generated in case errors are detected, regardless of the LIST values).

SCRATCH allows the user to specify a host directory spec for storing scratch files during TRASYS execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

TRASPLOT Execution - TRASPLOT provides interactive display of plots from a TRASYS generated plot file. The input plot file is specified by the user via an interactive prompt within the TRASPLOT module. There are no TRASPLOT RUN parameters.

3.4 SINDA/ISIN

This section describes the IAC implementation of the SINDA module (finite-difference thermal analyzer with fluid flow addition), and the associated IAC database interface module ISIN (Interface from SINda). Detailed SINDA documentation is contained in Reference 12. The general RUN schematic for executing these modules within ACE is shown in Figure 3.4-1. The following is a summary of SINDA RUN parameters which may be specified.

```
F        = basic file name
FIN      = input file spec
PRT      = print option                 (default = NO)
INTERVAL = save time interval
FOR      = user source names
OBJ      = user object names
LIB      = user library specs
LIST     = listing generation options   (default = NOFOR,NOCMP,NOLNK)
SCRATCH  = scratch directory spec       (default = installation defined)
```

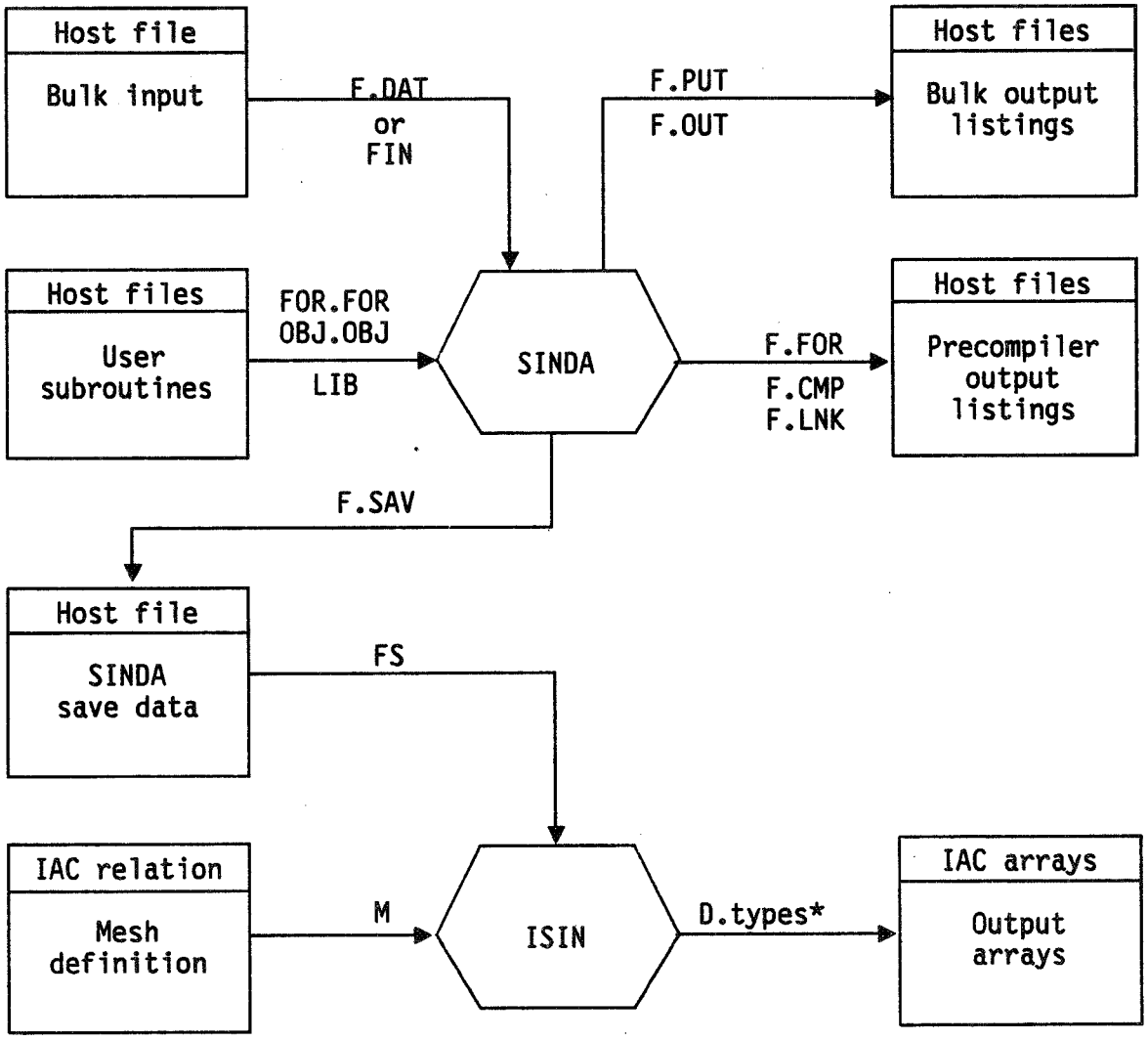
A common abbreviated form of the SINDA RUN statement is

```
RUN SINDA (F = name, PRT = option)
```

This statement executes a two-part SINDA module, consisting of a preprocessor to generate an executable processor program, and the processor to perform the analysis.

The F parameter in the RUN statement is used to name basic SINDA files. As shown in Figure 3.4-1, file type DAT is the bulk input file; PUT and OUT are output listing files; SAV is an optional output file containing data which may be inserted into an IAC database; and FOR, CMP and LNK are the respective FORTRAN, compiler and linker output files.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files



*types = TEMP, NCAP, COND,
QFLOW, QAPPL, QNET

```
RUN SINDA (F=name, FIN=spec, PRT=option, INTERVAL=interval+
          ,FOR=(name-list), OBJ=(name-list), LIB=(spec-list)+
          ,LIST=(option-list), SCRATCH=spec)

RUN ISIN  (FS=spec, M=spec, D=name, TYPE=(type-list))
```

Figure 3.4-1: SINDA/ISIN RUN Schematic

unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

The complete set of SINDA parameters is given by the RUN statement shown in Figure 3.4-1.

FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.

INTERVAL is an optional parameter; if given it causes output of certain SINDA computed data to the binary F.SAV file, and specifies the time interval at which that data is saved. The interval is relative to the SINDA print time step, e.g. a value of 1 indicates data saved at each print time step, 2 indicates every other print time step, etc. Selected data from the save file can later be processed and stored in an IAC database, via the interface module ISIN.

FOR, OBJ and LIB identify user subroutines in the form of source code, object code and object libraries, respectively. These subroutines are linked into the SINDA module prior to execution; in case of duplicate subroutine names, user routines will replace the corresponding SINDA routines. Note that each value of LIB defines a complete link type of spec which may include the directory and type and must include "/LIB".

LIST controls generation of the preprocessor output listings (printing is controlled by PRT); it is a keyvalue list which may contain FOR or NOFOR, CMP or NOCMP, and LNK or NOLNK, to specify generation or no generation of the respective FORTRAN, compiler and linker output files (compiler and linker files are generated in case errors are detected, regardless of the LIST values).

SCRATCH allows the user to specify a host directory for storing scratch files during SINDA execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

SINDA/IAC Data-Flow Approach - In developing the SINDA/IAC interface approach, a major consideration is the fact that the SINDA bulk input file may contain general user-supplied FORTRAN-like statements, which are compiled into executable code by a SINDA preprocessor. It is also considered desirable to have an open-ended SINDA/IAC data-flow approach, whereby additional data may in the future be identified for storage in the database. The developed approach therefore consists of the following steps.

1) The normal SINDA precompiler generates appropriate FORTRAN code, based on the user's bulk input file.

2) If the INTERVAL parameter has been specified, an IAC processor modifies the SINDA generated FORTRAN by inserting calls to a utility routine IACSAV. Each call sequence identifies a SINDA variable name (e.g. T) representing a vector of data, and a corresponding ISIN nametag, to be saved on the F.SAV file.

3) The SINDA link and execution steps are performed.

4) The interface routine ISIN may be run by the user, with a list of selected IAC array types. ISIN finds on the binary F.SAV file all nametag data variables required to create the selected arrays, creates the arrays, and inserts them into the IAC database.

This approach uses existing SINDA data handling tools and avoids modification of the SINDA code. It also provides a general capability by which additional SINDA/IAC data-flow can be implemented in the future.

ISIN Execution - The following is a summary of ISIN RUN parameters which may be specified.

FS	= SINDA save file spec	
M	= mesh relation spec	
D	= output array name	
TYPE	= output array types	(default = TEMP)

The form of the ISIN RUN statement is

```
RUN ISIN (FS = spec, M = spec, D = name, TYPE = (type-list))
```

The FS parameter gives the spec for the save file of SINDA computed data, to be input to ISIN.

M is an optional parameter which gives the spec of a mesh relation to be read from the database. If M is provided, the attributes (e.g. NODE,X,Y,Z) in this relation are incorporated, where appropriate, into the SINDA arrays output to the database.

D gives the name (or optionally the name:number) for all array files to be output by ISIN to the IAC database.

TYPE selects the array file types (TEMP, NCAP, COND, QFLOW, QAPPL, QNET) to be output to the IAC database. Default is the single array of type TEMP.

SINDA/IAC Arrays - The six currently available IAC array file types are described in Figure 3.4-2. Note that if the M parameter is specified, all occurrences of the attribute NID in the figure are replaced by the M mesh relation attributes.

An example RUN statement for ISIN might be

```
RUN ISIN (FS = SINPROB.SAV, D=PROBX, TYPE = (TEMP, QFLOW))
```

ISIN would then read the SINDA save file SINPROB.SAV, and create and insert the array files PROBX.TEMP and PROBX.QFLOW, into the IAC database.

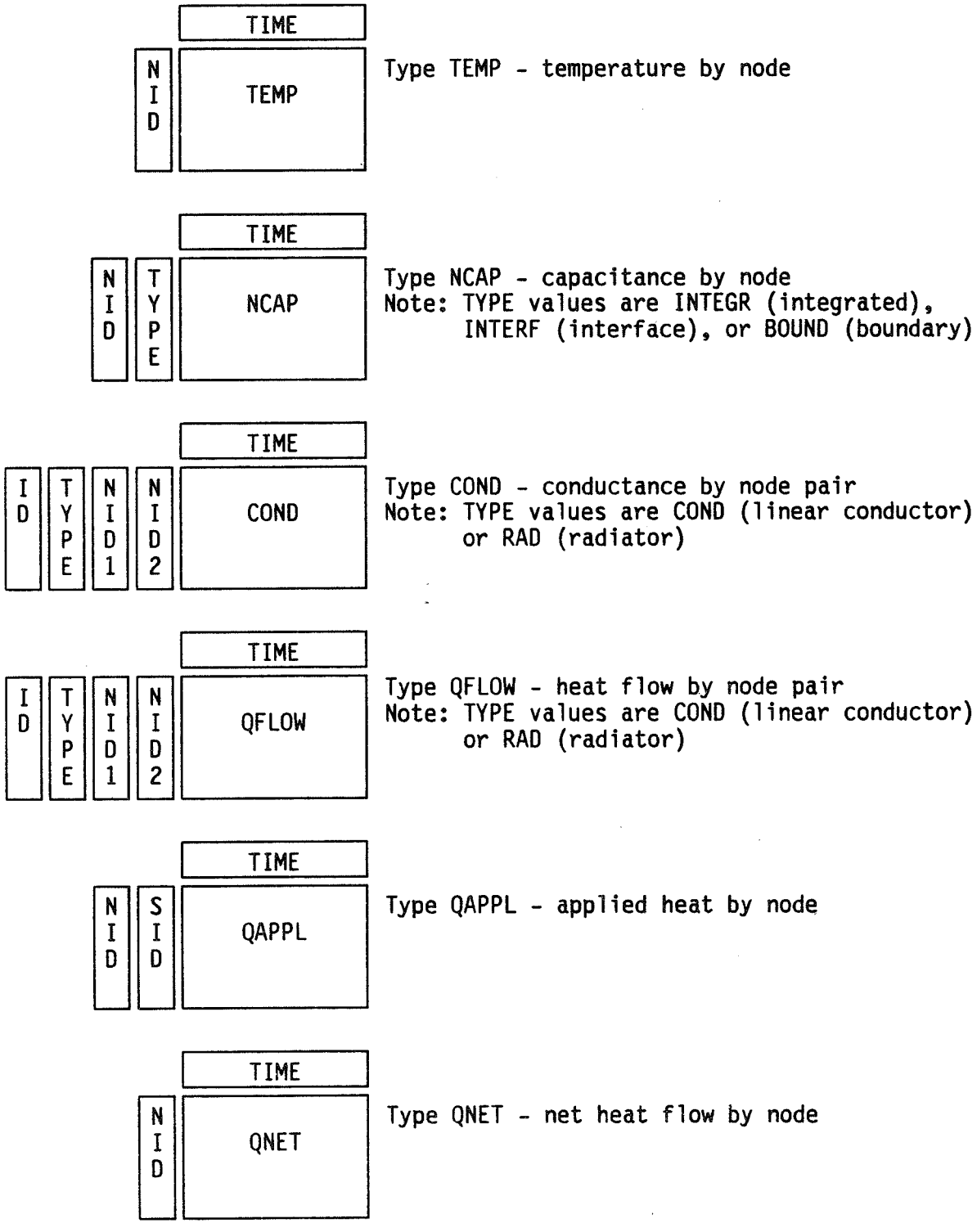


Figure 3.4-2: SINDA/IAC Arrays

Array type/description	Attribute name/type/description
TEMP - temperature by node	TEMP R1 - temperature values
	NID I1 - node id's
	TIME R1 - time values
NCAP - capacitance by node	NCAP R1 - capacitance values
	NID I1 - node id's
	TYPE C6 - node types
	TIME R1 - time values
COND - conductance by node pair	COND R1 - conductance values
	ID I1 - connector id's
	TYPE C4 - connector types
	NID1 I1 - node id 1
	NID2 I1 - node id 2
	TIME R1 - time values
QFLOW - heat flow by node pair	QFLOW R1 - heat flow values
	ID I1 - connector id's
	TYPE C4 - connector types
	NID1 I1 - node id 1
	NID2 I1 - node id 2
	TIME R1 - time values
QAPPL - applied heat by node	QAPPL R1 - applied heat values
	NID I1 - node id's
	SID I1 - source id's
	TIME R1 - time values
QNET - net heat flow by node	QNET R1 - net heat flow values
	NID I1 - node id's
	TIME R1 - time values

Figure 3.4-2: SINDA/IAC Arrays - (Continued)

3.5 ORACLS

ORACLS (References 13, 14) is a collection of subroutines for design of controllers and filters, emphasizing modern control methods. The general RUN schematic for executing the ORACLS module within ACE is shown in Figure 3.5-1, and details of the ORACLS capabilities and ORACLS/IAC interface are given in Appendix D. The following is a summary of ORACLS RUN parameters which may be specified.

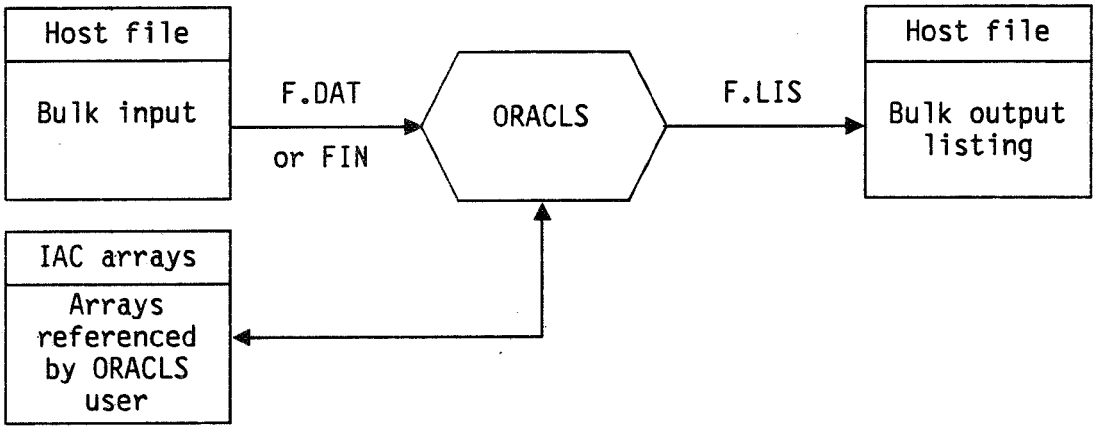
```
STD  = standard load module name     (default = REGEST)
EXE  = user load module spec
F    = basic file name
FIN  = input file spec
AREA = array storage area            (default = HD)
PRT  = print option                  (default = NO)
```

The form of the RUN statement is

```
RUN ORACLS (STD = name, EXE = name, F = name, FIN = spec,+
AREA = code, PRT = option)
```

which executes one of the standard or user defined ORACLS main programs.

STD and EXE are alternate (mutually exclusive) parameters, only one of which is given in a particular run. STD names a standard (IAC defined) program. The default STD value is REGEST which is a program to solve either the continuous or discrete time-invariant regulator/estimator/compensator problem with noise free measurements. The other standard driver program available is TIMFST to compute time responses of open and closed loop systems for discrete or continuous data. A description of these standard drivers and their operation is contained in Appendix D. The EXE parameter gives the spec for a special executable load module (user defined) program which may be created via the ACE LINK command.




```
RUN ORACLS (STD=name, EXE=name, F=name, FIN=spec, AREA=code+
           , PRT=option)
```

Figure 3.5-1: ORACLS RUN Schematic

The F parameter names the input file (name.DAT) and the output file (name.LIS). FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.

The AREA parameter gives the storage area from/to which IAC arrays are to be transferred. D indicates database; H indicates host directory; DH indicates database if possible, else host directory; and HD (default) indicates host directory if possible, else database. For example, an ORACLS array read operation following an AREA=DH specification results in an attempt to retrieve the array from the IAC database; if the retrieval is not successful (e.g. database not open, invalid IAC file spec, or file not found), an attempt is then made to retrieve the array from the host directory.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default), or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

3.6 MIMIC

MIMIC (Model Integration via Mesh Interpolation Coefficients) transforms field values (e.g. temperatures, pressures or stress tensors) from one model to another. Either scalar or nonscalar field values can be handled. The transformation involves coordinate based interpolation over 1-D, 2-D, or 3-D metric space. The coordinates are arbitrary, for example: time TIME; distance X; 2-D mesh coordinates X, Y; 3-D mesh coordinates X, Y, Z; etc.

This section describes the MIMIC RUN schematic and parameters, summarizes the transformation process and RUN cases, and provides some typical RUN examples. MIMIC theory and example problems are given in Appendix E.

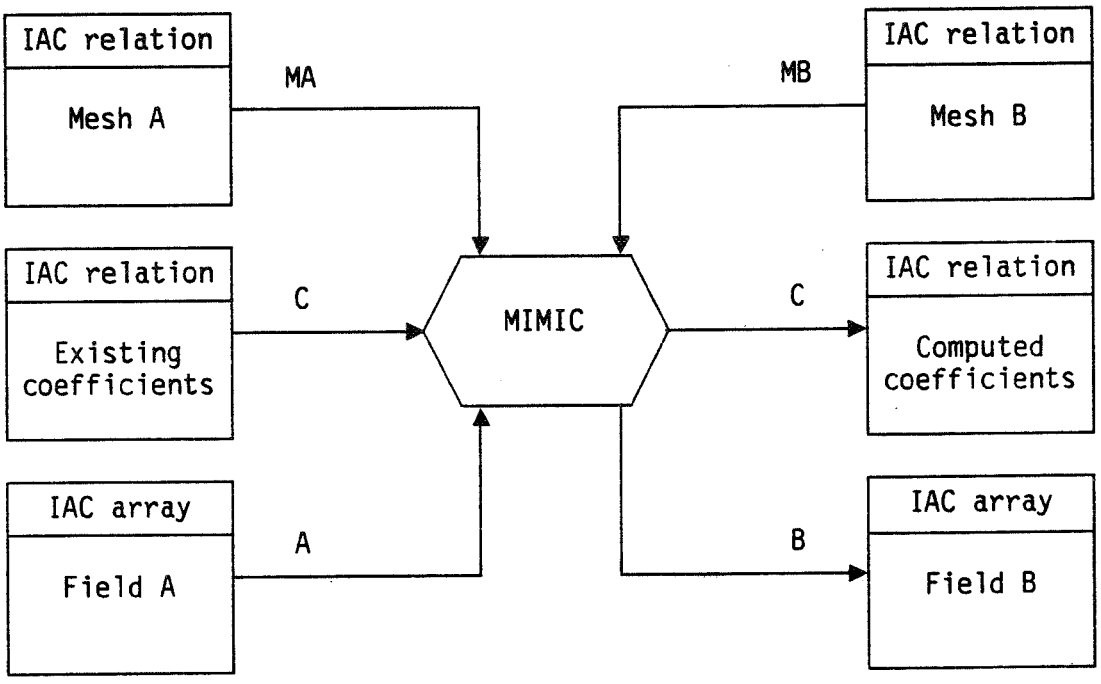
Run Schematic and Parameters - The general RUN schematic for executing the MIMIC module within ACE is shown in Figure 3.6-1. The following is a summary of MIMIC RUN parameters which may be specified.

MA	= mesh A spec	
MB	= mesh B spec	
C	= coefficients spec	
A	= field A spec	
B	= field B spec	
AREA	= data storage area	(default = HD)
XM	= mesh coordinate names	(default = X, Y, Z)
IMA	= identifier name in mesh A	(default = $I)
IFA	= identifier name in field A	(default = $I1)
EXE	= user load module spec	

A typical abbreviated form of the RUN statement is

```
RUN MIMIC (MA = spec, MB = spec, A = spec, B = spec)
```

In the naming convention for the above parameters, the characters A and B indicate entities associated with the respective meshes A and B.




```
RUN MIMIC (MA=spec, MB=spec, C=spec, A=spec, B=spec+
          ,AREA=area, XM=(name-list), IMA=name, IFA=name)
```

Figure 3.6-1: MIMIC RUN Schematic

The MA parameter gives the spec of the input mesh A relation. This relation must contain the IMA identifier attribute or index, whose values will be used to identify computed multipliers; it must also contain one or more of the XM coordinate attributes, whose values will be interpolated to compute the multipliers; it may in addition contain arbitrary other attributes.

MB gives the spec of the input mesh B relation. This relation must contain one or more of the XM coordinate attributes (these must be the same as those present in mesh A); it may in addition contain arbitrary other attributes, which will become part of the computed coefficients C relation and thereby passed on to become part of the field B array.

The C parameter gives the spec of the input or output coefficients relation, which contains the multiplier coefficients to be applied to field A values in order to compute field B values. Usually this relation is generated by MIMIC, and the user does not need to be concerned with its contents. It must include the attributes IDLIST and CFLIST; it may in addition contain arbitrary other attributes, which will become part of the computed field B array. The attribute IDLIST contains integer identifier values which match values of the IMA and IFA attributes; CFLIST contains double precision multiplier coefficients; if IDLIST and CFLIST are nonscalars, they must both have the same fixed size.

The A parameter gives the spec of the input field A array. This array must contain the IFA identifier attribute, whose values will be referenced by the coefficients IDLIST attribute.

B gives the spec of the output field B array to be computed.

The AREA parameter gives the storage area from/to which IAC relations and arrays are to be transferred. D indicates database; H indicates host directory; DH indicates database if possible, else host directory; and HD (default) indicates host directory if possible, else database. For example, a MIMIC mesh-A read operation following an AREA=DH specification results in an attempt to retrieve the mesh from the IAC database; if the retrieval is not successful (e.g. database not open, invalid IAC file spec, or file not

found), an attempt is then made to retrieve the mesh from the host directory.

The XM parameter may be used to specify names of mesh coordinates. Each coordinate must be of type real, either single or double precision. At least an initial sequence of these names must be present as attributes in both meshes A and B; for example, if XM = (X, Y, Z), then meshes A and B could both contain the attribute X or they could both contain the attributes X and Y. The mesh coordinates will be interpolated to compute values of the coefficients relation attributes IDLIST and CFLIST.

IMA may be used to give the name of the identifier in the mesh A relation. This identifier may be an attribute name (e.g. designating node id's); alternatively it may be the $I index name (e.g. designating node numbers). The identifier values must be of type integer.

IFA may be used to give the name of the identifier in the field A array. This identifier is similar to IMA; it may be either an attribute name, or one of the index names $I1, $I2, etc. The identifier values must be of type integer.

Note that more flexibility is provided if the IMA and IFA identifiers refer to an attribute rather than an index. The points in the field A array may then be ordered differently from those in the mesh A relation; in fact, the mesh A relation may even contain points which are not referenced by the field A array.

EXE is an optional parameter; if given it defines a complete spec for a user-defined load module, to be executed instead of the standard MIMIC module.

Transformation Process - The field value transformation process can be summarized by the symbolic equation

$$B = C \times A$$

which indicates that values in the field array B are computed by multiplying values in the field array A by coefficients in the relation C. As would be expected in such a multiplication process, there occurs a replacement of the

A array index and attributes associated with IFA, by the C relation index and attributes (except attributes IDLIST and CFLIST, which are discarded).

RUN Cases - MIMIC provides four basic RUN case options, as summarized by the following table.

Case	Parameters	Input	Output
1	MA, MB, C, A, B	MA, MB, A	C, B
2	MA, MB, A, B	MA, MB, A	B
3	MA, MB, C	MA, MB	C
4	C, A, B	C, A	B

Case 1 performs all steps in the transformation process. It uses the mesh relations MA and MB to compute the coefficients relation C, and stores C in the database or host directory. It then uses C and the field array A to generate the field array B, and stores B.

Case 2 is identical to Case 1, except that C is not stored.

Case 3 performs only the first part of the transformation process. It uses mesh relations MA and MB to compute the coefficients relation C, and stores C.

Case 4 performs only the last part of the transformation process. It uses the coefficients relation C and the field array A to generate the field array B, and stores B.

Typical RUN Examples - The RUN statement

```
RUN MIMIC (MA=A.MESH, MB=B.MESH, A=OLD.TEMP, B=NEW.TEMP)
```

performs a Case 2 process. Mesh coordinates are taken as either X, (X,Y) or (X, Y, Z), depending upon the attributes present in the A.MESH and B.MESH mesh relations. IMA is taken as $I and IFA is taken as $I1, indicating that all identifier definitions and references are based on index values. The

NEW.TEMP field array is computed and stored; it contains all attributes from OLD.TEMP, except that the attributes associated with index 1 are replaced by the attributes from B.MESH.

As another example, the RUN statement

```
RUN MIMIC (MA=THERMAL.MESH, MB=STRUCTURAL.MESH, C=C.DAT+
,IMA=NODE)
```

performs a Case 3 process. Mesh coordinates are taken in the same manner as for the previous example. The C.DAT coefficients relation is computed and stored. The NODE attribute in THERMAL.MESH is used as an identifier, which means that NODE values are stored in the C.DAT attribute IDLIST.

As a final example, the RUN statement

```
RUN MIMIC (MA=A.TIMES, MB=B.TIMES, A=PRES.A, B=PRES.B,+
C=C.STOR, XM=TIME, IFA='$I2')
```

performs a Case 1 process. The single mesh coordinate attribute TIME is used from A.TIMES and B.TIMES, i.e. interpolation is performed in a one-dimensional space. Index values from A.TIMES are stored in the C.DAT attribute IDLIST; these values then reference index 2 in the A.PRES field array. The B.PRES field array is computed and stored; it contains all attributes from A.PRES, except that attributes associated with index 2 are replaced by attributes from B.TIMES.

3.7 INTA

The IAC interface module INTA (Interface from NASTRAN Thermal Analyzer) reads data from a NASTRAN OUTPUT2 file, created by either an IACDMAP2 nonlinear steady-state or an IACDMAP3 transient thermal run. INTA reformats the data into IAC standard array form, and stores the arrays in the IAC database.

The general RUN schematic for executing the INTA module within ACE is shown in Figure 3.7-1. The following is a summary of INTA RUN parameters which may be specified.

FN = NASTRAN OUTPUT2 file spec
D = output array name
TYPE = output array types

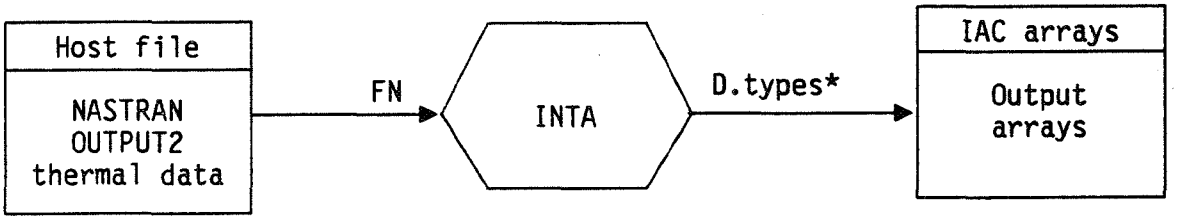
The form of the INTA RUN statement is

RUN INTA (FN = name, D = name, Type = (type-list))

The FN parameter gives the spec for the NASTRAN OUTPUT2 file, to be input to INTA.

This file is generated during the NASTRAN IACDMAP2 and IACDMAP3 execution, and the user does not usually need to be concerned with its contents. It contains the NASTRAN GPLS table of node (grid and scalar point) id's; the BGPDTS table of node location coordinates (basic system); the ECTS table of element connections; the EST element summary table; the BJJ matrix of nodal capacitances; the OLB1 table of output times (only for a transient run); the UGVS matrix of nodal temperatures; and the OEF1X table of element flux related data.

The D parameter gives the name (or optionally the name:number) for all array files to be output by INTA to the IAC database.



* types=TEMP, NCAP, FLUX

RUN INTA (FN=spec, D=name, TYPE=(type-list))

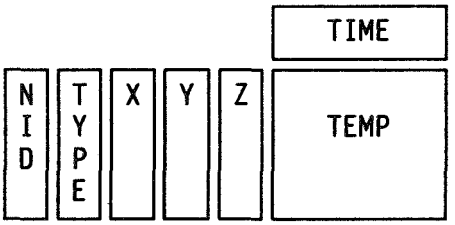
Figure 3.7-1: INTA RUN Schematic

TYPE selects the array file types (TEMP, FLUX, NCAP) to be output to the IAC database.
An example run statement for INTA might be

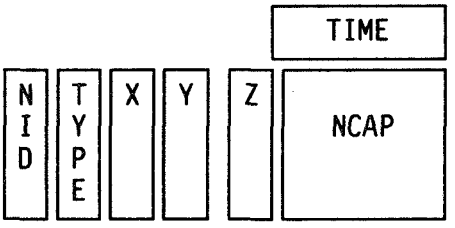
```
RUN INTA (FN = NASPROB3.SAV, D=PROB3, TYPE = (FLUX, TEMP))
```

INTA would then read the NASTRAN OUTPUT2 file NASPROB3.SAV, and create and insert the array files PROB3.FLUX and PROB3.TEMP, into the IAC database.

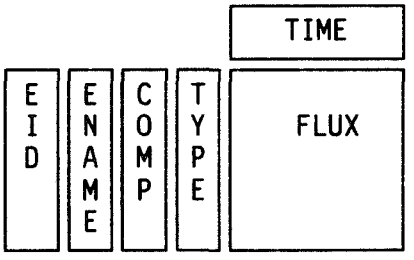
NASTRAN-Thermal/IAC Files - The three currently available IAC array file types are described in Figure 3.7-2.



Type TEMP - nodal temperatures
Note: TYPE values for NASTRAN grid points are G; for scalar points are S.



Type NCAP - nodal capacitances
Note: TYPE values for NASTRAN grid points are G; for scalar points are S.



Type FLUX - element flux data
Note: COMP values for CHBDY element are QAPPL, QCONV, QRAD, QTOTAL; for other elements are XGRAD, YGRAD, ZGRAD, XFLUX, YFLUX, ZFLUX

TYPE values for CHBDY element are AREA3, AREA4, LINE, ELLCYL, POINT, FTUBE, REV; for other elements are blank.

Figure 3.7-2: NASTRAN-Thermal/IAC Arrays

Array type/description	Attribute name/type/description
TEMP - nodal temperature	TEMP R1 - nodal temperatures
	NID I1 - node id's
	TYPE C1 - node types
	X R1 - X coordinates
	Y R1 - Y coordinates
	Z R1 - Z coordinates
	TIME R1 - time values
NCAP - nodal capacitance	NCAP R2 - nodal capacitances
	NID I1 - node id's
	TYPE C1 - node types
	X R1 - X coordinates
	Y R1 - Y coordinates
	Z R1 - Z coordinates
	TIME R1 - time values
FLUX - element flux data	FLUX R1 - element fluxes or temperature gradients
	EID I1 - element id's
	ENAME C8 - element names
	COMP C6 - flux or gradient components
	TYPE C6 - element types
	TIME R1 - time values

Figure 3.7-2: NASTRAN-Thermal/IAC Arrays (Continued)

3.8 INDA

The IAC interface module INDA (Interface from NASTRAN Dynamics Analyzer) reads data from a NASTRAN OUTPUT2 file created by an IACDMAP1 normal-modes analysis, reformats it into IAC standard relation and/or array files, and stores the files in the database.

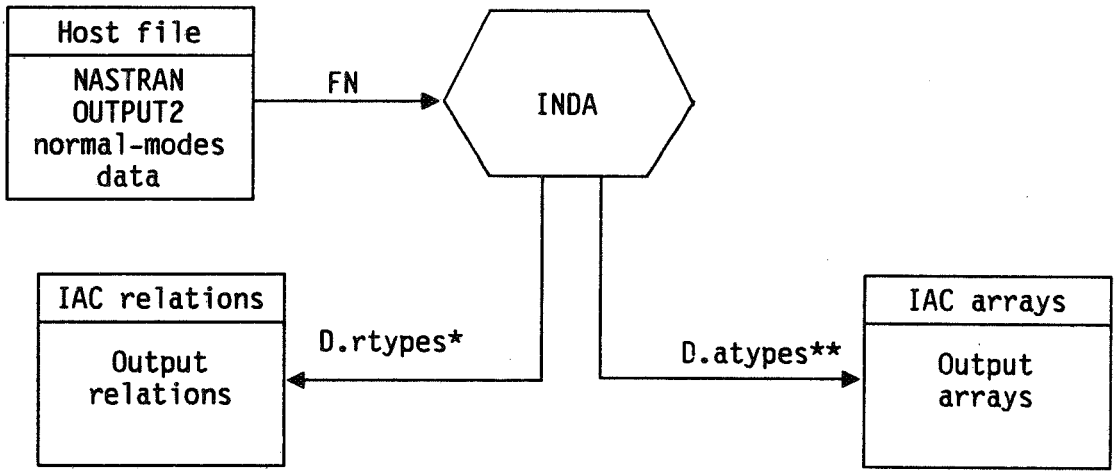
INDA is usually executed within IAC Solution Path III or before IAC Solution Path IV (see Section 4). The general RUN schematic for executing the INDA module within ACE is shown in Figure 3.8-1. The following is a summary of INDA RUN parameters which may be specified.

```
FN   = NASTRAN OUTPUT2 file spec
MODE = mode id's
D    = output file name
TYPE = output file types
```

The FN parameter gives the spec for the NASTRAN OUTPUT2 file to be input to INDA. This file was generated during the IACDMAP1 execution, and normally the user is not concerned with its contents. It contains the NASTRAN GPL table of node id's; the BGPDT table of node location coordinates (basic system); the g-set mass matrix, MGG; the g-set eigenvector matrix, PHIG; and the modal stiffness, mass and damping matrices KHH, MHH and BHH. The BHH matrix is optional.

MODE specifies a list of mode id values, which may optionally include the IAC range operator ".." and the IAC increment operator "&&". If an increment is not defined, 1 is the default. An example mode id list could be of the form "(1, 3 6 .. 10, 30 .. 60 && 10)". MODE is required when file types MODE, MSTIF, MDAMP and/or MMASS are output to the database.

The D parameter gives the name (or optionally the name:number) for all IAC relation and/or array files output by INDA to the IAC database.



* rtypes = NODE, NMASS
** atypes = MODE, MSTIF, MDAMP, MMASS

RUN INDA (FN=spec, MODE=(mode-id-list), D=name, TYPE=(type-list))

Figure 3.8-1: INDA RUN Schematic

TYPE selects the file types (NODE, NMASS, MODE, MSTIF, MDAMP and MMASS) to be output to the IAC database. The default is to select all types for which data exists on the OUTPUT2 file.

An example RUN statement for INDA might be

```
RUN INDA (FN=TETRA.SAV, MODE = (1..5)
    , D=TETRA , TYPE=(NODE,NMASS,MODE,MSTIF))
```

INDA would read the NASTRAN OUTPUT2 file TETRA.SAV and create and insert IAC files TETRA.NODE, TETRA.NMASS, TETRA.MODE and TETRA.MSTIF into the database. TETRA.MSTIF would contain modal stiffnesses for only modes 1, 2, 3, 4 and 5.

NASTRAN-Dynamics/IAC Files - the available IAC file types are described in Figure 3.8-2. Types NODE and NMASS are relations and the others are arrays.

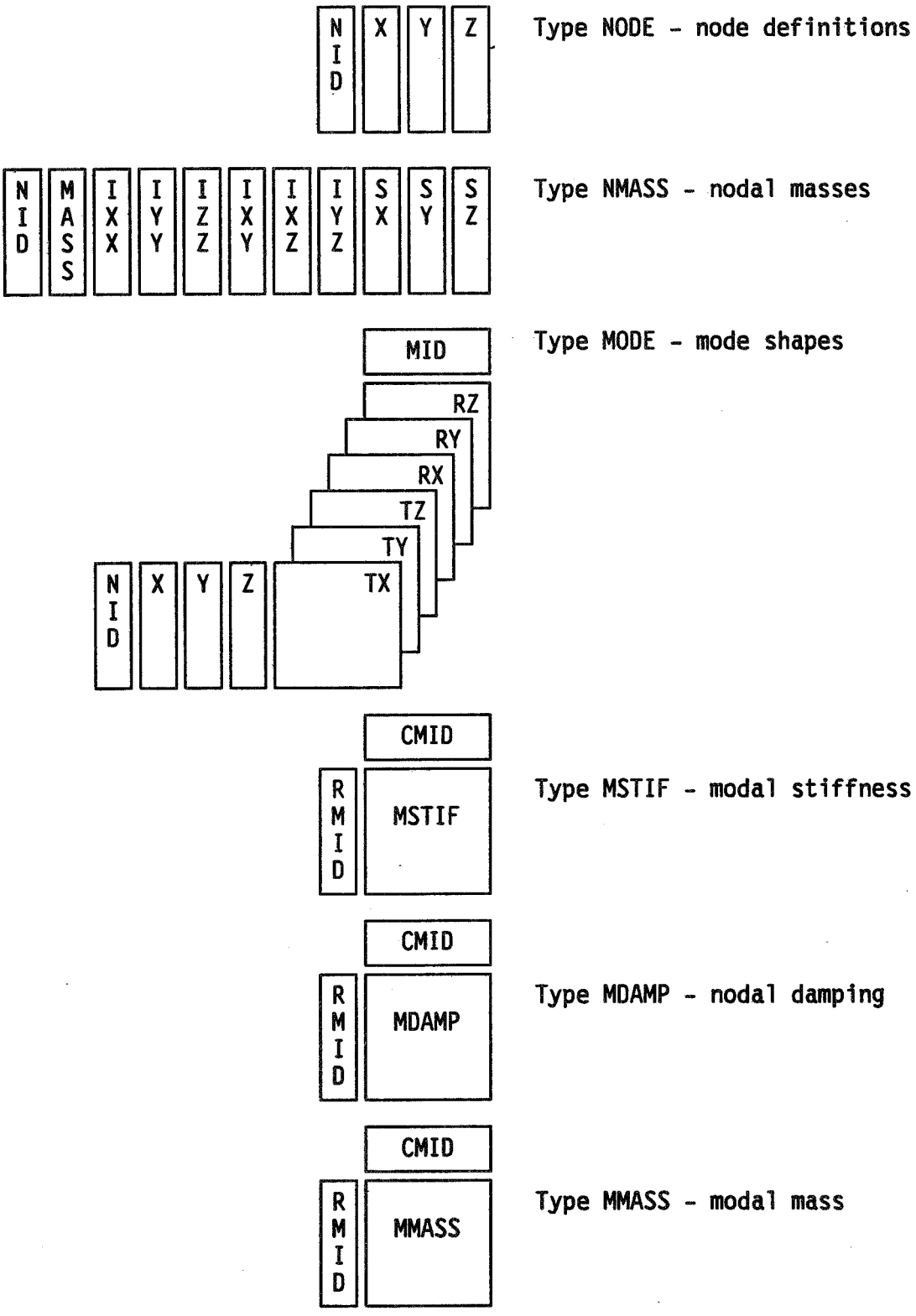


Figure 3.8-2: NASTRAN-Dynamics/IAC Files

File type/description	Attribute name/type/description		
NODE - node definitions	NID	I1 -	node id's
	X	R1 -	X coordinates
	Y	R1 -	Y coordinates
	Z	R1 -	Z coordinates
NMASS - nodal masses	NID	I1 -	node id's
	MASS	R2 -	masses
	IXX	R2 -	XX inertias
	IYY	R2 -	YY inertias
	IZZ	R2 -	ZZ inertias
	IXY	R2 -	XY inertias
	IXZ	R2 -	XZ inertias
	IYZ	R2 -	YZ inertias
	SX	R2 -	X static moments
	SY	R2 -	Y static moments
	SZ	R2 -	Z static moments
MODE - mode shapes	TX	R2 -	nodal X translations
	TY	R2 -	nodal Y translation
	TZ	R2 -	nodal Z translation
	RX	R2 -	nodal X rotations
	RY	R2 -	nodal Y rotations
	RZ	R2 -	nodal Z rotations
	NID	I1 -	node id's
	X	R1 -	X coordinates
	Y	R1 -	Y coordinates
	Z	R1 -	Z coordinates
	MID	I1 -	mode id's

Figure 3.8-2: NASTRAN-Dynamics/IAC Files (Continued)

File type/description	Attribute name/type/description
MSTIF - modal stiffness	MSTIF R2 - modal stiffnesses
	RMID I1 - row mode id's
	CMID I2 - column mode id's
MDAMP - modal damping	MDAMP R2 - modal damping
	RMID I1 - row mode id's
	CMID I1 - column mode id's
MMASS - modal mass	MMASS R2 - modal masses
	RMID I1 - row mode id's
	CMID I1 - column mode id's

Figure 3.8-2: NASTRAN-Dynamics/IAC Files (Continued)

3.9 INSAT

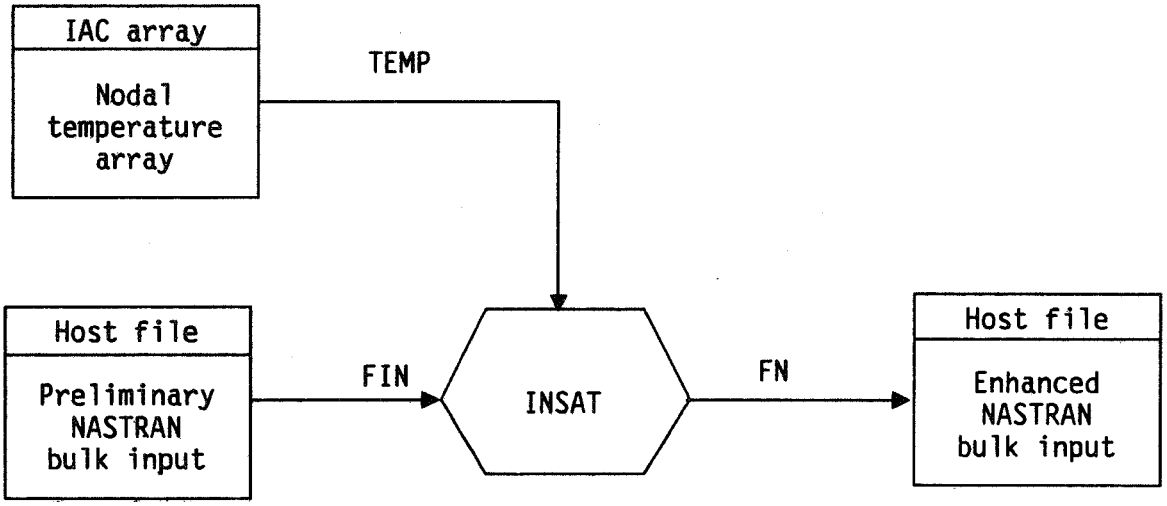
The IAC interface module INSAT (Interface to NASTRAN Statics Analyzer for Thermal data) reads a preliminary NASTRAN bulk input file for a structural statics run, and an array of nodal temperature data. INSAT then creates an enhanced NASTRAN bulk input file by automatically generating cards which define thermal load sets, and a table which correlates time values with solution subcase id's.

INSAT is usually executed prior to the IACDMAP5 static solution sequence (see Section 3.1.2), and within the IAC Solution Paths II and IV (see Section 4). The NASTRAN statics run will include thermal load efects, however, the preliminary user-supplied input file is missing the required thermal load sets.

INSAT generates the NASTRAN thermal load sets, one for each time specified by the user in the INSAT RUN command. INSAT also generates a NASTRAN DTI table named IACTIME which correlates the specified time values with solution subcase id's. The times are ordered as specified in the RUN command. INSAT merges these data with the preliminary input file, and creates an enhanced input file suitable for NASTRAN IACDMAP5 execution. Note that the IACTIME table is passed along by the NASTRAN IACDMAP5 sequence to an OUTPUT2 file, where it is utilized by the INSA module (see Section 3.11) to form certain IAC database arrays.

The general RUN schematic for executing the INSAT module within ACE is shown in Figure 3.9-1. The following is a summary of INSAT RUN parameters which may be specified.

FIN	=	preliminary NASTRAN input file spec	
TEMP	=	input array spec	
TIME	=	time value list	
FN	=	enhanced NASTRAN input file spec	
SCID	=	beginning subcase id	(default = 1001)
TLID	=	beginning thermal load set id	(default = 1001)




```
RUN INSAT (FIN=spec, TEMP=spec, TIME=(time-list)+
          ,FN=spec, SCID=id, TLID=id)
```

Figure 3.9-1: INSAT RUN Schematic

A common form of the RUN statement for INSAT is

RUN INSAT (FIN = spec, TEMP = spec, TIME = (time-list), FN = spec)

The FIN parameter gives the spec of the preliminary user-supplied NASTRAN bulk input file, read by INSAT.

TEMP gives the spec of the input nodal temperature array, which is read from the IAC database. The array file must contain a TEMP attribute of real temperatures. The temperature values are in the form of a 2-D array (number of nodes by number of times). The array file must also contain an NID attribute of integer node id's, associated with array index 1; and a TIME attribute of real times, associated with array index 2. The file may in addition contain arbitrary other attributes. Note that each real attribute may be of either single or double precision type.

TIME specifies a list of real time values, which may optionally include the IAC range operator ".." and the IAC increment operator "&&". Note that with real values, use of one of these operators requires the other operator. For example, the time list could be of the form "(1., 2.0, 6...20. && 2.0)". For each specified time value, nodal temperatures are extracted via interpolation from the array, and a corresponding thermal load set is inserted into the NASTRAN input file. INSAT provides linear interpolation on time, and values of the array TIME attribute must be in increasing order; if a specified time value is outside the range of the array, temperatures are taken from the closest time available, and a warning message is written.

The FN parameter gives the spec of the enhanced NASTRAN bulk input file, written by INSAT.

SCID is an optional parameter which specifies the beginning subcase id (in the IACTIME table). The default value is 1001; i.e. subcases are assigned default id's of 1001, 1002, ---, 1000+n, corresponding to the list of n specified times.

TLID Is an optional parameter which specifies the beginning thermal load set id. The default value is 1001; i.e. thermal load sets are assigned default id's of 1001, 1002,---, 1000+n, corresponding to the list of n specified times.

In order to provide desired flexibility, the user is given certain options and responsibilities. The user is responsible for generating all required NASTRAN case-control data, including solution subcase definitions which select the INSAT generated thermal load sets. In performing this task the user should maintain consistency of the data, with the thermal load set id's and IACTIME table subcase id's generated by INSAT; this can be accomplished by defining user data consistent with INSAT generated data, or by editing the INSAT data. The user may choose to supplement the INSAT thermal load definitions with mechanical loads or additional thermal loads. Then it is the user's responsibility to assign appropriate additional load case id's and solution subcase id's, and to add appropriate data to the case-control and the IACTIME table.

An example RUN statement for INSAT might be

```
RUN INSAT(FIN = DISHP.DAT, TIME = (0.2, 0.8, 1.4, 5.0)+
, TEMP = TLOAD1.TEMP, FN = DISHC.DAT, SCID = 101)
```

INSAT would then read the preliminary NASTRAN bulk input file DISHP.DAT, and extract nodal temperatures (at time points 0.2, 0.8, 1.4, 5.0) from the IAC database array TLOAD1.TEMP. For these respective time points it would generate NASTRAN thermal load sets (with set id's 1001, 1002, 1003, 1004); and the IACTIME table (correlating the selected time values and the subcase id's 101, 102, 103, 104). The generated data would be merged with existing card images in the DISHP.DAT file, to create the enhanced NASTRAN bulk input file DISHC.DAT. Assuming the existence of appropriate data in the DISHP.DAT file, the DISHC.DAT file would then be ready for use in a NASTRAN statics run to compute nodal displacements, stresses, etc.

3.10 INSAM

The IAC interface module INSAM (Interface to NASTRAN Statics Analyzer for Modal data) reads a preliminary NASTRAN bulk input file for a structural statics run, and an array of mode-shape definitions. INSAM then creates an enhanced NASTRAN bulk input file by automatically generating cards which define a nodal-to-modal conversion; the conversion definition includes scalar-point freedoms, multi-point constraints, and a table which correlates mode id's with scalar-point id's. The theoretical basis for the INSAM nodal-to-modal conversion is described in Appendix F.

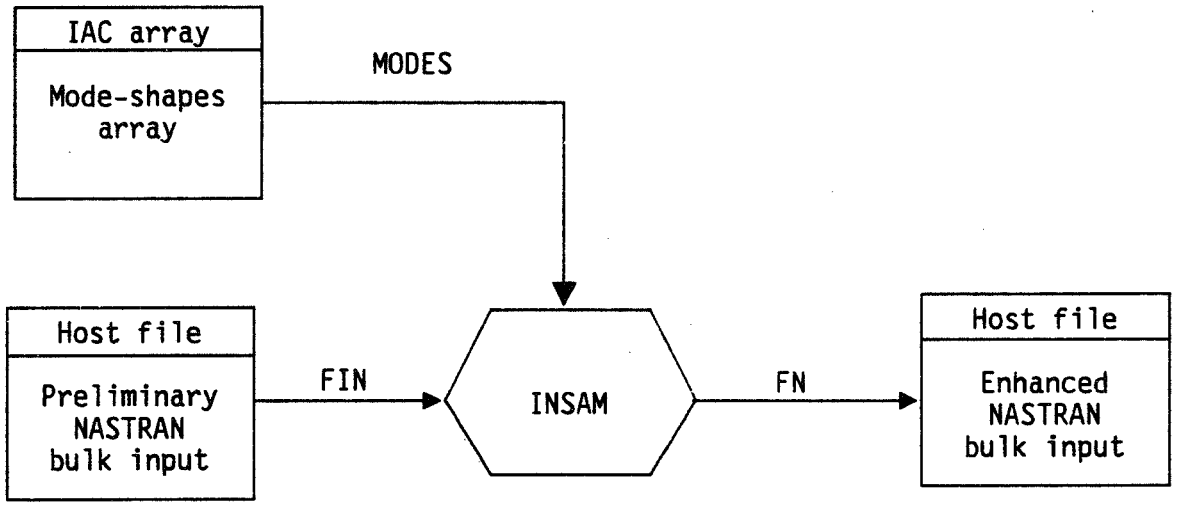
INSAM is usually executed prior to the IACDMAP5 sequence (see Section 3.1.2) and within the IAC Solution Path IV (see Section 4). The NASTRAN statics run will be based on modal displacement freedoms, however, the preliminary user-supplied input file is missing the nodal-to-modal conversion definition.

INSAM generates NASTRAN scalar-point freedoms, one for each mode; and MPC equations, one for each existing nodal freedom. The form of the MPC equations is

$$q = \phi a$$

where q is a particular nodal freedom, a is the vector of scalar freedoms, and ϕ is a row of the total mode-shapes matrix. INSAM also generates a NASTRAN DTI table named IACMODE which correlates the mode id's with scalar-point id's. The mode id's are ordered as given in the input mode-shape arrays. INSAM merges these data with the preliminary input file, and creates an enhanced input file suitable for NASTRAN IACDMAP5 execution. Note that the IACMODE table is passed along by the NASTRAN IACDMAP5 sequence to an OUTPUT2 file, where it is utilized by the INSA module (see Section 3.11) to form certain IAC database arrays.

The general RUN schematic for executing the INSAM module within ACE is shown in Figure 3.10-1. The following is a summary of INSAM RUN parameters which may be specified.




```
RUN INSAM (FIN=spec, MODES=spec, FN=spec,SPID=id, MPCID=id+
          ,CONT=value)
```

Figure 3.10-1: INSAM RUN Schematic

FIN	=	preliminary NASTRAN input file spec	
MODES	=	input array spec	
FN	=	enhanced NASTRAN input file spec	
SPID	=	beginning scalar-point id	(default = 1001)
MPCID	=	MPC set id	(default = 1001)
CONT	=	beginning MPC continuation	(default = 1001)

A common form of the RUN statement for INSAM is

```
RUN INSAM (FIN = spec, MODES = spec, FN = spec)
```

The FIN parameter gives the spec of the preliminary user-supplied NASTRAN bulk input file, read by INSAM.

MODES gives the spec of the input mode-shapes array, which is read from the IAC database. The array file must contain one or more of the six attributes T1, T2, T3, R1, R2, R3 of real nodal displacements, which respectively correspond to the X-Y-Z translations and X-Y-Z rotations; the displacement values for each component are in the form of a 2-D array (number of nodes by number of modes); any of the six attributes not present in the file is treated by INSAM as if it contained all zero values. The array file must also contain an NID attribute of integer node id's, associated with array index 1; and an MID attribute of integer mode id's, associated with array index 2. The file may in addition contain arbitrary other attributes. Note that each real attribute may be of either single or double precision type.

The FN parameter gives the spec of the enhanced NASTRAN bulk input file, written by INSAM.

SPID is an optional parameter which specifies the beginning scalar-point id. The default value is 1001; i.e. scalar points are assigned default id's of 1001, 1002, ---, 1000+m, corresponding to the list of m input modes.

MPCID is an optional parameter which specifies the multi-point constraint set id. The default id is 1001.

CONT is an optional parameter which specifies an integer used to form the beginning MPC card continuation indicator. The default integer is 1001; i.e. MPC card continuation indicators are assigned values B001001, B001002, ---, in sequence as required.

The user is responsible for generating all required NASTRAN case-control data, including selection of the INSAM generated MPC set. The case control will not generally include other MPC or SPC set selections, unless the user wishes to define a subcase for which the INSAM nodal-to-modal conversion is not performed. However, any rigid body modes should be constrained via SUPPORT cards.

An example RUN statement for INSAM might be

```
RUN INSAM (FIN=BOOM.DAT, MODES=BMODES.DAT+
,FN=BOOMM.DAT, SPID=101, CONT-5001)
```

INSAM would then read the preliminary NASTRAN bulk input file BOOM.DAT, and extract modal data for all existing modes from the database array file BMODES.DAT. It would generate NASTRAN scalar points with id's 101, 102, ---, 100+m); the IACMODE table (correlating the mode id's and the generated scalar-point id's); and MPC equations (using the MPC set id 1001, and continuation indicators B005001, B005002, ---). The generated data would be merged with existing card images in the BOOM.DAT file, to create the enhanced NASTRAN bulk input file BOOMM.DAT. Assuming the existence of appropriate data in the BOOM.DAT file, the BOOMM.DAT file would then be ready for use in a NASTRAN statics run to compute nodal and modal displacements, stresses, etc.

3.11 INSA

The IAC interface module INSA (Interface from NASTRAN Statics Analyzer) reads data from a NASTRAN OUTPUT2 file created by an IACDMAP5 structural statics run, reformats it into IAC standard array and/or relation files, and stores the files in the IAC database. The general RUN schematic for executing the INSA module within ACE is shown in Figure 3.11-1. The following is a summary of INSA RUN parameters which may be specified.

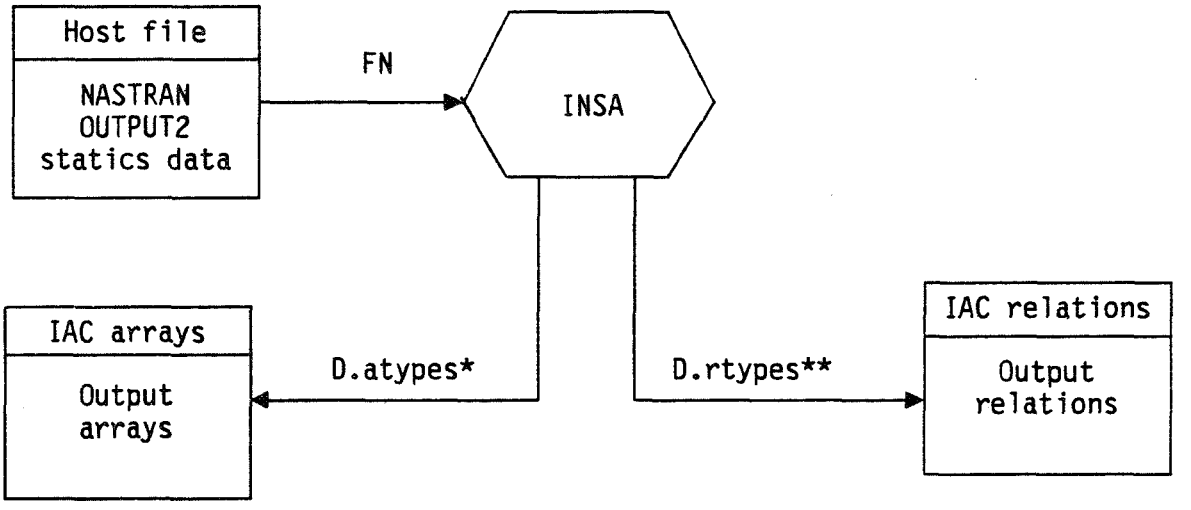
```
FN     =  NASTRAN OUTPUT2 file spec
D      =  output file name
TYPE   =  output file types
```

The form of the INSA RUN statement is

```
RUN INSA (FN = spec, D = name, TYPE = (type-list))
```

The FN parameter gives the spec for the NASTRAN OUTPUT2 file, to be input to INSA. This file is generated during the NASTRAN IACDMAP5 execution, and the user does not usually need to be concerned with its contents. It contains the NASTRAN CASECC table of Case Control; the GPL table of node (and mode, i.e. scalar point) id's; the BGPDT table of node location coordinates (basic system); and the UGV matrix of nodal displacements. It may also contain the IACTIME table which correlates time values with solution subcase id's (see module INSAT); the IACMODE table which correlates mode id's with scalar point id's, thereby partitioning the set of scalar point freedoms into nodal freedoms and modal freedoms (see module INSAM); the ONRGY1 table of element energy data; and the OGPFB1 table of node force balance data.

The D parameter gives the name for all array and relation files to be output by INSA to the IAC database. The number (in the IAC spec) for these files is equal to 1 for the arrays, and is equal to the NASTRAN subcase id for the relations. Note that each IAC array contains data for all subcases, while each IAC relation contains data for only a single subcase.



*atypes = NDISP, MDISP
**rtypes = ENERGY, NFORCE, NSTRESS, NSTRAIN

RUN INSA (FN=spec, D=name, TYPE=(type-list))

Figure 3.11-1: INSA RUN Schematic

Type selects the array and relation file types (NDISP, MDISP, ENERGY, NFORCE, NSTRESS, NSTRAIN), to be output to the IAC database.

NASTRAN-Statics/IAC Database Files - The six currently available IAC file types are described in Figure 3.11-2. The first two types are arrays, and the last four types are relations.

The MDISP array can be generated only if the IACMODE table is available, in which case MDISP contains data for only the tabulated scalar points (modes); the SPID and MID attribute values are ordered as in the table.

If the IACTIME table is not available, the NDISP and MDISP arrays contain data for all subcases referenced by the NASTRAN output requests. If this table is available, NDISP and MDISP contain data for only the tabulated subcases (times); the subcase and TIME attribute values are ordered as in the table.

The ENERGY and NFORCE relations can be generated only if the respective ESE and GPFORCE output requests were present in the NASTRAN Case Control. Note that these relations contain data for only the elements and nodes referenced by the NASTRAN output requests. A separate relation is generated for each subcase referenced by the output requests. Additional information on these data is available in the NASTRAN Programmer Manual, Section 2.3.79.

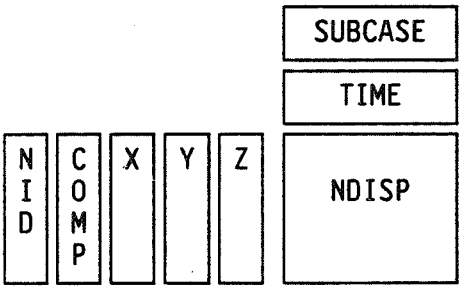
The NSTRESS relation can be generated only if both the GPSTRESS and STRESS output requests were present in the NASTRAN Case Control. Note that this relation contains data for only those nodes/elements defined on SURFACE and/or VOLUME Case Control cards. A separate relation is generated for each subcase referenced by the output requests. Additional information on these data is available in the NASTRAN User Manual, Section 3.2.4; and the Programmer Manual, Sections 2.3.129 and 4.206.

The NSTRAIN relation has form and Case Control requirements similar to the NSTRESS relation, except that the STRESS output request is replaced by STRAIN.

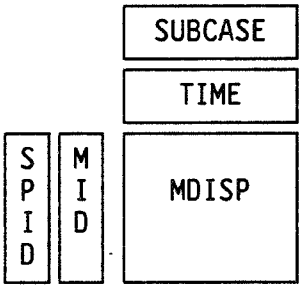
An example RUN statement for INSA might be

```
RUN INSA (FN=STATRUN.SAV, D=STATBEAM, TYPE=(NDISP,ENERGY))
```

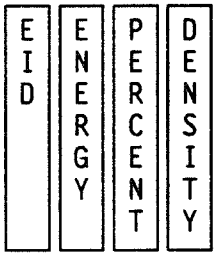
INSA would then read the NASTRAN OUTPUT2 file STATRUN.SAV and create and insert the array file STATBEAM.NDISP and one or more relation files STATBEAM.ENERGY, into the IAC database.



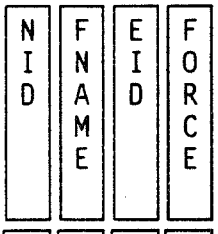
Type NDISP - nodal displacements
Note: COMP values for NASTRAN Grid points are T1, T2, T3, R1, R2, R3; for single scalar points are S.



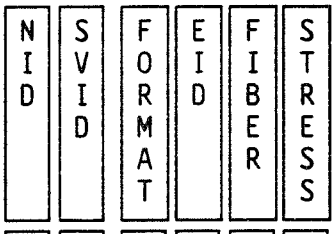
Type MDISP - modal displacements



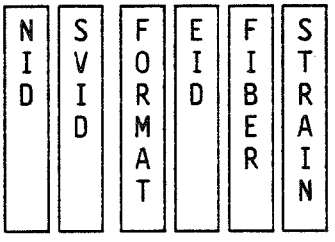
Type ENERGY - element strain energy



Type NFORCE - nodal force balance



Type NSTRESS - nodal stresses
Note: FORMAT values are S for surface stress; VD for volume direct stress; VP for volume principal stress. Fiber values are Z1, Z2, or MID.



Type NSTRAIN - nodal strains
Note: FORMAT values are S for surface stress; VD for volume direct stress; VP for volume principal stress. Fiber values are MID or CURV.

Figure 3.11-2: NASTRAN-Statics/IAC Files

File type/description	Attribute name/type/description		
NDISP - nodal displacements	NDISP	R2	- nodal displacements
	NID	I1	- node id's
	COMP	C2	- node components
	X	R1	- X coordinates
	Y	R1	- Y coordinates
	Z	R1	- Z coordinates
	SUBCASE	I1	- subcase id's
	TIME	R1	- time values
MDISP - modal displacements	MDISP	R2	- modal displacements
	SPID	I1	- scalar point id's
	MID	I1	- mode id's
	SUBCASE	I1	- subcase id's
	TIME	R1	- time values
ENERGY - element strain energy	EID	I1	- element id's
	ENERGY	R1	- strain energies
	PERCENT	R1	- percent of totals
	DENSITY	R1	- energy densities
NFORCE - nodal force balance	NID	I1	- node id's
	FNAME[*]	C8	- force names
	EID[*]	I1	- element id's
	FORCE[6,*]	R1	- forces

Figure 3.11-2: NASTRAN-Statics/IAC Files (Continued)

File type/description	Attribute name/type/description		
NSTRESS - nodal stresses	NID	I1	- node id's
	SVID	I1	- surface or volume id's
	FORMAT	C2	- stress formats
	EID	I1	- element id's
	FIBER	C4	- stress locations
	STRESS[*]	R1	- nodal stresses
NSTRAIN - nodal strains	NID	I1	- node id
	SVID	I1	- surface or volume id
	FORMAT	C2	- strain format
	EID	I1	- element id
	FIBER	C4	- strain type
	STRAIN[*]	R1	- nodal strains

Figure 3.11-2: NASTRAN-Statics/IAC Files (Continued)

3.12 ACE

ACE (Analysis Capability Executive) is the main executive program for IAC. ACE provides several major functions:

1. an input command/tutorial processor, including on-line help and examples;

2. a driver code to execute other general application modules;

3. an access controller to allow concurrent usage of databases;

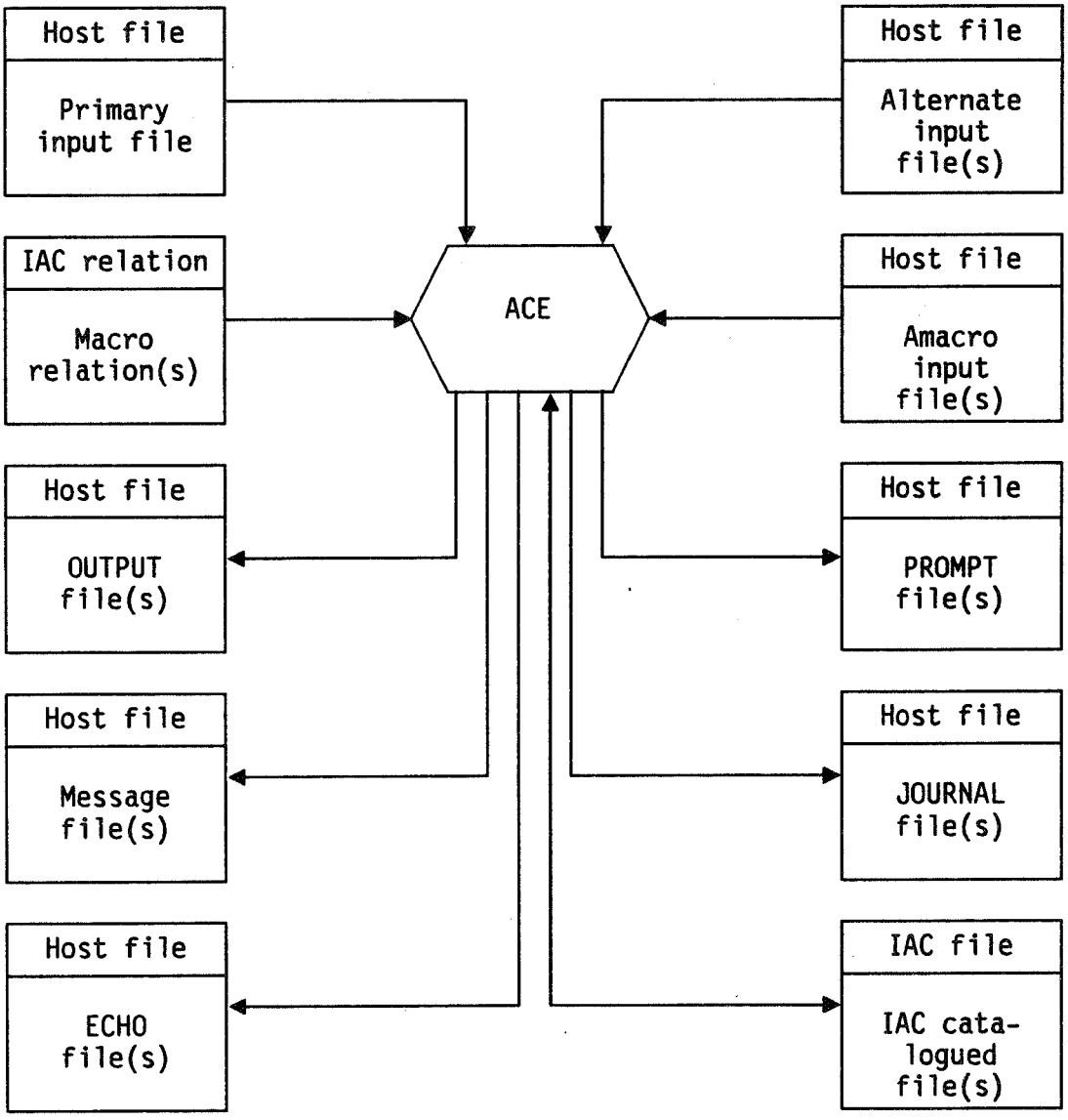
4. an extensive data management/query capability, including data-structures of relational, array and user-defined forms, and providing cataloging of both structured and unstructured files; and

5. a generic data processor, supporting an interface between IAC and "foreign module" character-formatted data.

ACE executes in the same general manner as any other IAC module. For example, ACE itself may be executed via a RUN command within the ACE executive. (Although not usually done, such an execution might be useful, e.g. to make available to the user two different executive workspaces). The general RUN schematic for executing ACE is shown in Figure 3.12-1. The following is a summary of ACE RUN parameters which may be specified.

```
INPUT = input strings
EXE   = user load module spec
```

The INPUT parameter is optional and gives the ACE input stream, in the form of a list of one or more strings. If given, each string value becomes a line in the ACE primary input file. If not given, the ACE primary input file is the terminal for an interactive run or a null file for a batch run. (An attempt to read a null file results in an end-of-file error message.)



RUN ACE (INPUT=(string-list), EXE=spec)

Note: At start of execution, the three input type host files are all equivalenced to one another; the output, prompt and message files are equivalenced; and the JOURNAL and ECHO files are absent. The ACE "SET IO" command may be used to redefine the file specs and/or file equivalences.

Figure 3.12-1: ACE RUN Schematic

EXE is an optional parameter; if given it defines a complete spec for a user-defined load module, to be executed instead of the standard ACE module.

3.13 PLOT

PLOT is the IAC system plot module. It is used to create graphs and charts, which generally display relationships between a number of different variables. PLOT can handle an arbitrary number of variables, points, curves, legends, etc.

The tabular data for PLOT usually originates from an IAC relation or array, in which each variable is represented as an attribute and each point is represented as a tuple. However, the input to PLOT is in the form of a character formatted plot file. This file may be created within the IAC ACE executive, via the PLOT command; it may also be created directly by a user or an application module.

Appendix B defines the plot file format. The general RUN schematic for the PLOT module is shown in Figure 3.13-1. There are no RUN parameters; plot file specs may be selected interactively within the PLOT module.

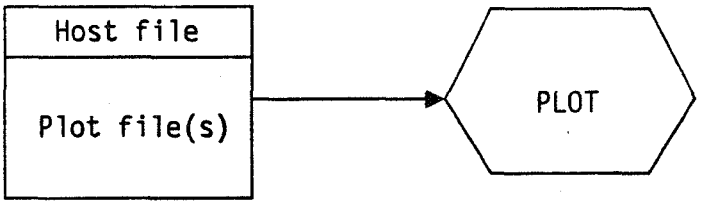


Figure 3.13-1: PLOT RUN Schematic

3.14 MODELI

This section describes the IAC implementation of the MODELI (Multi Optimal Differential Equation Language, Interactive dialect) module for generating and displaying numerical solutions of ordinary differential equations. Detailed MODELI documentation is contained in Reference 17. The general RUN schematic for executing the MODELI module within ACE is shown in Figure 3.14-1. The following is a summary of MODELI RUN parameters which may be specified.

```
FM       = new model file name
FA       = analysis file name
PRT      = print option                  (default = NO)
EXE      = executable load module name
FOR      = user source names
OBJ      = user object names
LIB      = user library specs
LIST     = listing generation options    (default = NOFOR,NOCMP,NOLNK)
SCRATCH  = scratch directory spec        (default = installation defined)
```

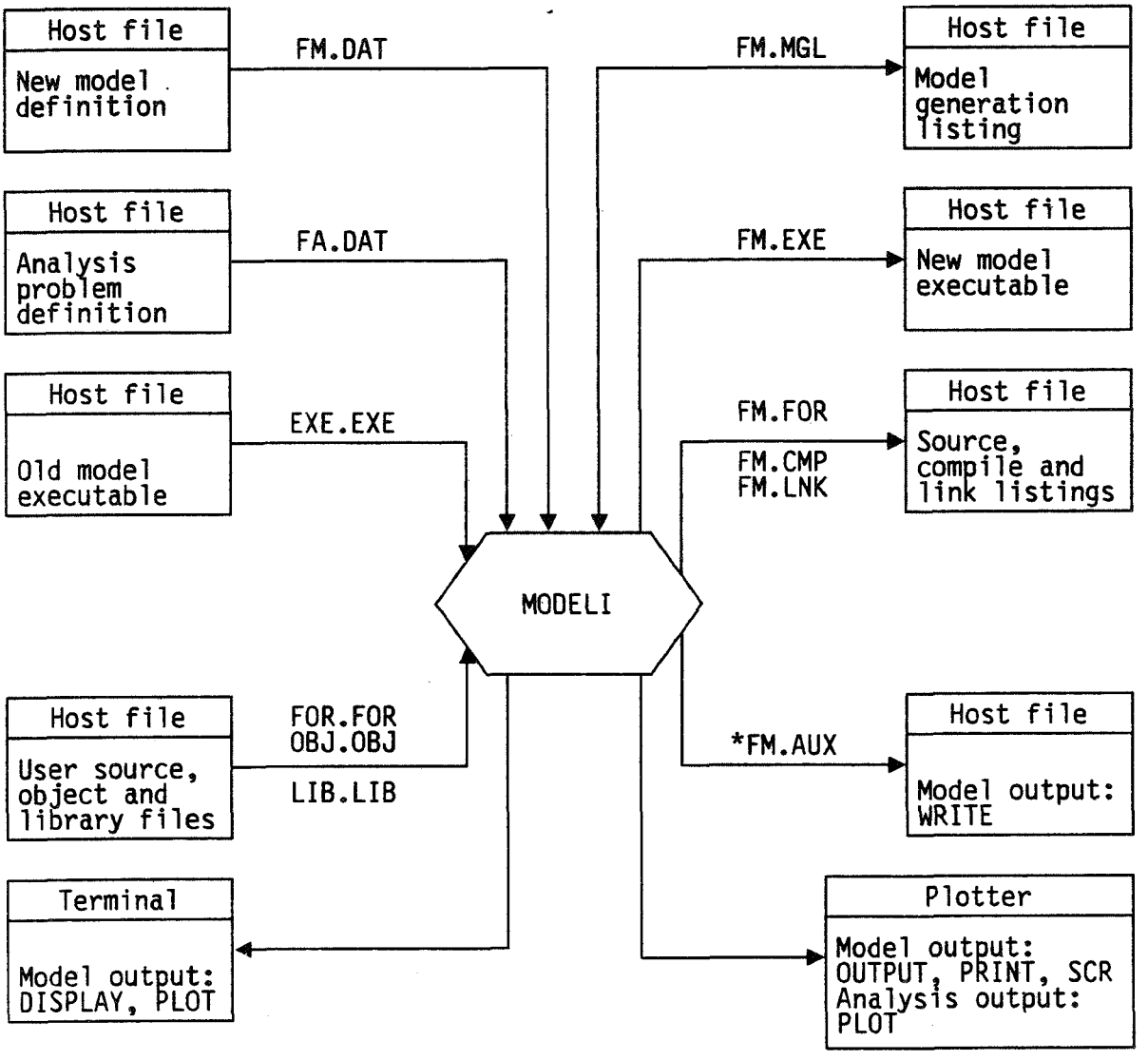
A common abbreviated form of the MODELI RUN statement is

```
RUN MODELI (FM = name, PRT = option)
```

This statement executes a two-part MODELI module, consisting of a preprocessor to generate an executable processor program, and the processor to perform the analysis.

The FM parameter is used to name files associated with the MODELI preprocessor. File type DAT is the new model definition input file; MGL is the model generation listing; EXE is the new model executable processor program; and FOR, CMP and LNK are the respective FORTRAN, compiler and linker output files. File type AUX is output from the MODELI WRITE command.

FA is used to name files associated with the MODELI processor. File type DAT is the analysis problem definition input file; and AUX is output from the MODELI WRITE command (if the FM parameter is not defined).



* If FM parameter is not given, spec is FA.AUX
If FM and FA are not given, spec is EXE.AUX

```
RUN MODELI (FM=name, FA=name, PRT=option, EXE=name+
           ,FOR=(name-list), OBJ=(name-list), LIB=(spec-list) +
           ,LIST=(option-list), SCRATCH=spec)
```

Figure 3.14-1: MODELI RUN Schematic

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

EXE is used to save the new model executable processor program, when FM is defined. IF FM is not defined, EXE is the name of the executable processor program from an old model.

FOR, OBJ and LIB identify user subroutines in the form of source code, object code and object libraries, respectively. These subroutines are linked into the MODELI processor module prior to execution; in case of duplicate subroutine names, user routines will replace the corresponding MODELI routines. Note that each value of LIB defines a complete link type of spec which may include the directory and type and must include "/LIB".

LIST controls generation of the preprocessor output listings (printing is controlled by PRT); it is a keyvalue list which may contain FOR or NOFOR, CMP or NOCMP, and LNK or NOLNK, to specify generation or no generation of the respective FORTRAN, compiler and linker output files (compiler and linker files are generated in case errors are detected, regardless of the LIST values).

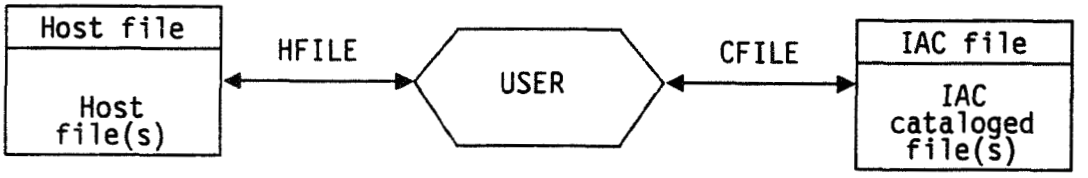
SCRATCH allows the user to specify a host directory for storing scratch files during MODELI execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

3.15 USER

This section describes the IAC implementation of the USER module. USER provides a generic capability for executing a user owned module within ACE, without requiring the usual module implementation procedure described in Section 7. For example, the information usually inserted into the MODULES.MOD and module RUN tables is already provided for USER in a generic form. Note, however, that if RUN parameters are to be passed from ACE to the user module, the user module must be programmed to read these parameters (see Section 7). The USER capability may be convenient for initial testing of new modules, and for handling of modules which are not often used or which are specific to only a few users. The following is a summary of USER RUN parameters which may be specified.

```
EXE      = executable file spec of module software
JCL      = command file spec of module software
I1       = integer values
R1       = real values
R2       = double precision real values
Z2       = complex values
Z4       = double precision complex values
L1       = logical values
C        = character values
HFILE    = host file specs
CFILE    = IAC cataloged file specs
NAME     = name values
NUMBER   = number values
NAMNUM   = name:number values
TYPE     = type values
STRING   = string values
AR       = arithmetic values
NA       = nonarithmetic values
```

EXE and JCL are alternate (mutually exclusive) parameters, only one of which is given in a particular run. EXE gives the spec of the user module executable file, and JCL gives the spec of the user module command file. One




```
RUN USER (EXE=spec, JLC=spec+
          ,I1=(I1-list), R1=(R1-list), R2=(R2-list)+
          ,Z2=(Z2-list), Z4=(Z4-list), L1=(L1-list)+
          ,C=(C-list), HFILE=(HFILE-list), CFILE=(CFILE-list)+
          ,NAME=(NAME-list), NUMBER=(NUMBER-list)+
          ,NAMNUM=(NAMNUM-list), TYPE=(TYPE-list)+
          ,STRING=(STRING-list), AR=(AR-list), NA=(NA-list))
```

Figure 3.15-1: USER RUN Schematic

of these parameters is required in order to accomplish execution of the user module software.

The other parameters are all optional, and provide for various types of values to be transferred to the user module, via the IAC parameter file. See Sections 6.4 and 7.0 for information on the parameter file, and Table 2.1-1 for additional description of the data types. These parameters are summarized as follows.

I1 defines a list of integer values, which may optionally include the IAC range operator ".." and the IAC increment operator "&&".

R1 defines a list of real values, which may optionally include the IAC range operator ".." and the IAC increment operator "&&".

R2 defines a list of double precision values, which may optionally include the IAC range operator ".." and the IAC increment operator "&&".

Z2 defines a list of complex values. Each complex value is composed of a real value and an imaginary value separated by blank(s) and/or a comma.

Z4 defines a list of double precision complex values. Each complex value is composed of a double precision real value and a double precision imaginary value separated by blank(s) and/or a comma.

L1 defines a list of logical values.

C defines a list of character values (each value is of arbitrary length).

HFILE defines a list of host file specs.

CFILE defines a list of IAC cataloged file specs.

NAME defines a list of alphanumeric values. Each value must be composed of a maximum of 10 characters and the first character must be alpha. Alpha is A-Z and "__".

NUMBER defines a list of integer values. Each value must be composed of a maximum of 10 integer characters.

NAMNUM defines a list of values of the form name:number. The name part must be of type NAME and the number part must be of type NUMBER. The :number part is optional.

Type defines a list of character values. Each value must be I1, R1, R2, Z2, Z4, L1, Cn or C*, where n is an unsigned integer.

STRING defines a list of character values. Each character value must be enclosed with apostrophes.

AR defines a list of arithmetic values. Each value must be readable by a single FORTRAN F format.

NA defines a list of nonarithmetic values. Each value must be enclosed in apostrophes or not be readable by a single FORTRAN F format.

Each given value of the parameters I1, R1, etc. results in the sequential generation of one logical card image record on the USER module parameter file (or more than one record if the value is greater than 50 characters in length). An exception occurs for the complex parameters Z2 and Z4, where each value generates two records (the first for the real part and the second for the imaginery). For example, the RUN statement

```
RUN USER  (EXE=MYPROG.EXE, I1=(1,10,100), R1=23.5+
          , Z2=(1.0,2.0  5.E6,3.5E5), C=FLAG3)
```

would generate consecutive parameter file records containing the following value fields. (See Section 7 for other fields on the records.)

```
1
10
100
23.5
1.0
2.0
5.E6
3.5E5
FLAG3
```

3.16 RIM

RIM (Relational Information Management system) is a database management module. RIM is based upon the relational algebra model for data management and has been used for both engineering and business data. The module may be executed in either of two modes: menu or command. The menu mode prompts the user for the input required to create, update and/or query the database. The command mode requires the direct input of commands. A guide to using RIM can be found in Reference 18. The general RUN schematic for executing RIM within ACE is shown in Figure 3.16-1. The IAC implementation of RIM provides the following optional RUN parameters.

```
F   = basic file name
FIN = input file spec
PRT = print option      (default = NO)
```

A common form of the RUN statement for an interactive execution is

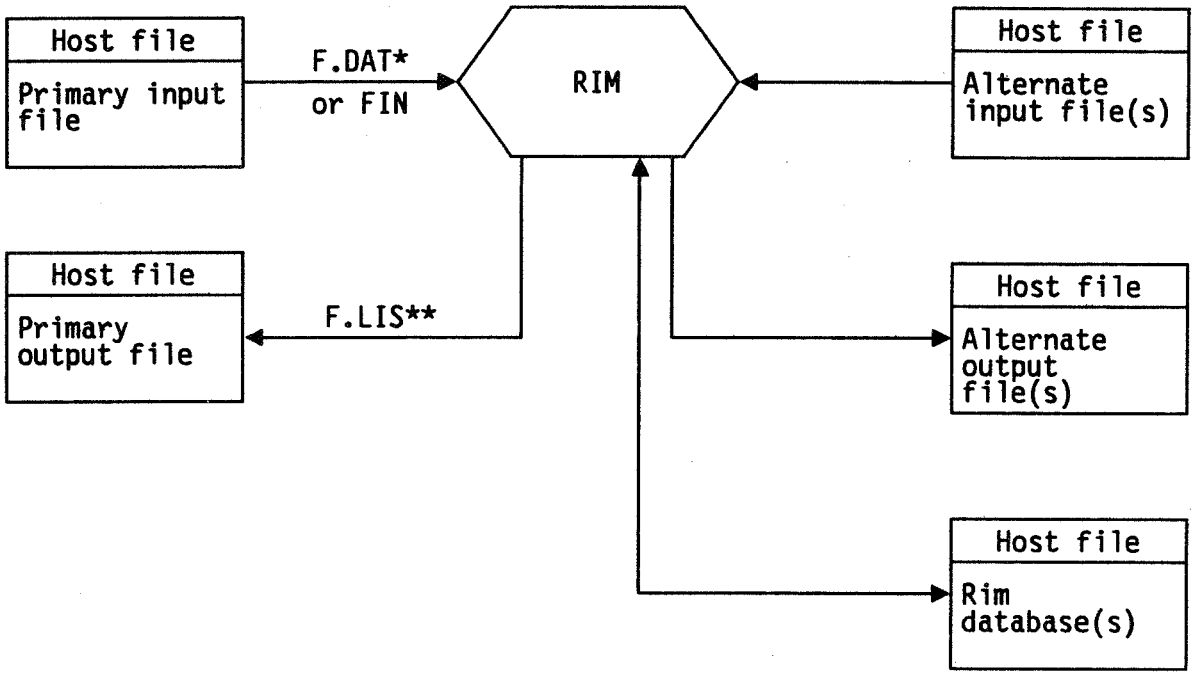
```
RUN RIM
```

which assumes the primary input and output files are assigned to the terminal.

The optional F parameter names the primary input file (name.DAT) and the primary output file (name.LIS). If F is not defined, the primary input file is the terminal (interactive) or a null file (batch), and the primary output file is the terminal (interactive) or a log file (batch).

FIN is an optional parameter which defines a complete input file spec to be used instead of F-spec name.DAT.

The PRT parameter controls printing of the output listing file, and equals one of the keyvalues YES, NO (default), or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print



RUN RIM (F=name, FIN=spec, PRT=option)

* If neither F nor FIN is given, the terminal is the primary input file.

** If F is not given, the terminal is the primary output file.

Figure 3.16-1: RIM RUN Schematic

queue, with later printing by the user followed by automatic deletion from the user's directory.

IAC/RIM Data Transfer - Data can be transferred from IAC to RIM by using IAC's SET IO and PRINT commands and RIM's LOAD and INPUT commands. The data transfer from RIM to IAC is accomplished by RIM's OUTPUT and UNLOAD commands and IAC's GETG command. The vehicle of transfer is a formatted host file which is written by IAC and read by RIM, or written by RIM and read by IAC. In many cases the file does not have to be edited before it is read.

To minimize any editing, the following IAC and host subset capabilities should be emphasized.

Data structure classes	relation only
Attribute names	1-8 alphanumeric characters
Attribute types	scalars, and 1- or 2-dimensional nonscalars
Data types	integer, real, double precision and character
Host file specs	1-8 characters (prefer alphanumeric name only)
Host file record length	80 characters

IAC-to-RIM Example - Assume that the B-mesh coordinates as defined in Figure E.2-2 exist as an IAC relation in the form of a host file. The following sequence of ACE and RIM commands results in the storing of this relation into a RIM database, and the display of the relation by RIM. Note that if the IAC relation had existed in an IAC database, the first ACE command GETS would be replaced by the command GET.

```
ACE >GETS  MESH.B  MESH.B
ACE >SET IO  OUTPUT=TORIM
ACE >PRINT MESH.B  $A  *
ACE >RUN RIM
R>OPEN TEST
R>DEFINE TEST
```

```
D>ATTRIBUTES
D>POINT INT
D>X  DOUB
D>Y  DOUB
D>RELATION
D>MESHB WITH POINT X Y
D>END
R>LOAD MESHB
L>INPUT TORIM
L>END
R>SELECT ALL FROM MESHB
```

RIM-To-IAC Example - Assume that the B-mesh coordinates from the previous example exist as a RIM relation in a RIM database. The following sequence of ACE and RIM commands results in the storing of this relation into an ACE workspace, and the display of the relation by ACE. Note that the relation could then be stored permanently via the ACE PUT or PUTS command.

```
ACE >RUN RIM
R>OPEN TEST
R>OUTPUT TOIAC
R>UNLOAD DATA MESHB
R>OUTPUT OUTPUT
R>EXIT
ACE >HOST 'EDIT TOIAC'
       (delete RIM non data lines at beginning and at end of file TOIAC)
ACE >GETG  TOIAC  MESHB.FROMRIM  +
      POINT,I2  ,X,R2  ,Y,R2
ACE >PRINT MESHB.FROMRIM  $A  *
```

Note on Character Data - ACE and RIM use an apostrophe and a double quote, respectively, as enclosing delimiters for character data. The delimiter is required if the data contains an embedded blank or special character. An embedded delimiter must be doubled. The ACE PRINT command does not automatically enclose character data with delimiters, but can be made to add

a beginning and trailing double quote to an attribute value via appropriate definition of the PRINT format. If POINT is a character data attribute, then the appropriate ACE command would be

```
ACE >REDEFINE MESH.B  POINT,C10/'1H"  ,A, 1H"'
```

RIM's UNLOAD command always encloses character data in double quotes. The quotes have to be converted to apostrophes (if required) or eliminated by an editor before input to the ACE GETG command.

Note on Non-Scalars - RIM input requires one level of parentheses for vectors and two levels for matrices. The ACE PRINT command does not automatically enclose non-scalars with parentheses, but can be made to add a left and right parentheses by adding parentheses literals to the PRINT-items list.

As an illustration, if the X and Y coordinates from the previous example were defined in the form of a 2 by n matrix XY, then the PRINT command would be

```
ACE > PRINT MESH.B POINT,'(',XY[..],')' *
```

RIM's UNLOAD command always uses parentheses to define vectors and matrices. GETG does not accept parentheses, thus these must be eliminated (e.g. by editing) before GETG is executed.

For other special operations involving non-scalars, the ACE PRINT dimensionality function $D, and cardinality functions $C and $Ci, may also be useful in transferring information about the data.

3.17 SAMSAN

SAMSAN (Reference 19) is an expandable library of state-of-the-art algorithms for classical analysis of large order systems emphasizing SAMpled System ANalysis. The general RUN schematic for executing the SAMSAN module within ACE is shown in Figure 3.17-1. The following is a summary of SAMSAN RUN parameters which may be specified.

```
STD  =  standard load module name  (default  =  NULL)
EXE  =  user load module spec
F    =  basic file name
FIN  =  input file spec
AREA =  array storage area         (default  =  HD)
PRT  =  print option               (default  =  NO)
```

The form of the RUN statement is

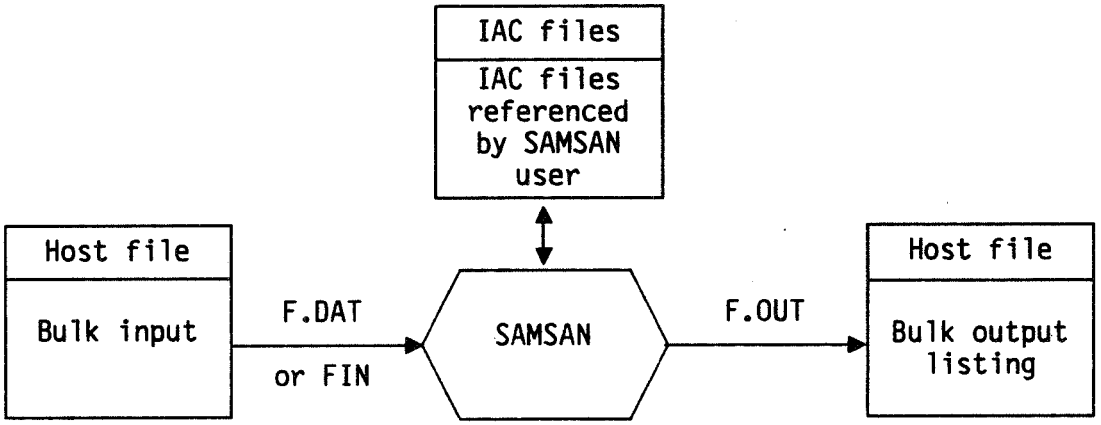
```
RUN SAMSAN (STD = name, EXE = name, F = name, FIN = spec,+
AREA = code, PRT = option)
```

which executes one of the standard or user defined SAMSAN main programs.

STD and EXE are alternate (mutually exclusive) parameters, only one of which is given in a particular run. STD names a standard (IAC defined) program. The default STD value is NULL, which prints the parameter file on the terminal (or log file for batch). The EXE parameter gives the spec for a special executable load module (user defined) program which may be created via the ACE LINK command.

The F parameter names the input file (name.DAT) and the output file (name.OUT). FIN is an optical parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.

The AREA parameter gives the storage area from/to which IAC arrays are to be transferred. D indicates database; H indicates host directory; DH indicates database if possible, else host directory; and HD indicates host directory if




```
RUN SAMSAN (STD=name, EXE=name, F=name, FIN=spec, AREA=code+
           , PRT=option)
```

Figure 3.17-1: SAMSAN RUN Schematic

possible, else database. For example, a SAMSAN array read operation following an AREA=DH specification results in an attempt to retrieve the array from the IAC database; if the retrieval is not successful (e.g. database not open, invalid IAC file spec, or file not found), an attempt is then made to retrieve the array from the host directory.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default), or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

4.0 SOLUTION PATHS

In this section some important multi-discipline technical capabilities of IAC are described in terms of four predefined solution paths, for which particular computational and data-flow requirements have been identified and implemented. These solution paths are: I) Standalone; II) Thermal/Structural; III) Structural/Control (time or frequency domain); and IV) Thermal/Structural/Control (time domain). A Solution Path V Thermal/Structural/Control (fully coupled) has been studied and is a possible future extension, applicable to problems such as thermal flutter. (See Appendix F.4). The solution paths are summarized in Figure 4.0-1.

A modular philosophy has been emphasized in developing these IAC solution path capabilities; that is, the various computational and data-flow operations have been implemented as individual logical building blocks within IAC, to be combined by the engineering analyst as required for a particular application. This modular engineer-in-the-loop approach is used in preference to the alternative of a more monolithic and highly automated process. It is felt that the needs of complex design analysis tasks are best served by a detailed involvement on the part of the analyst, in order to 1) develop an insight into the system behavior, and 2) use this insight in a creative manner to make required analytical decisions. The solution paths are therefore described as complete sequences, but are composed of well defined individual execution steps. You can run a complete solution path following the steps indicated, rerun certain portions of a path, or modify some of the solution steps to suit particular user requirements. The following sections address each of the respective IAC solution paths.

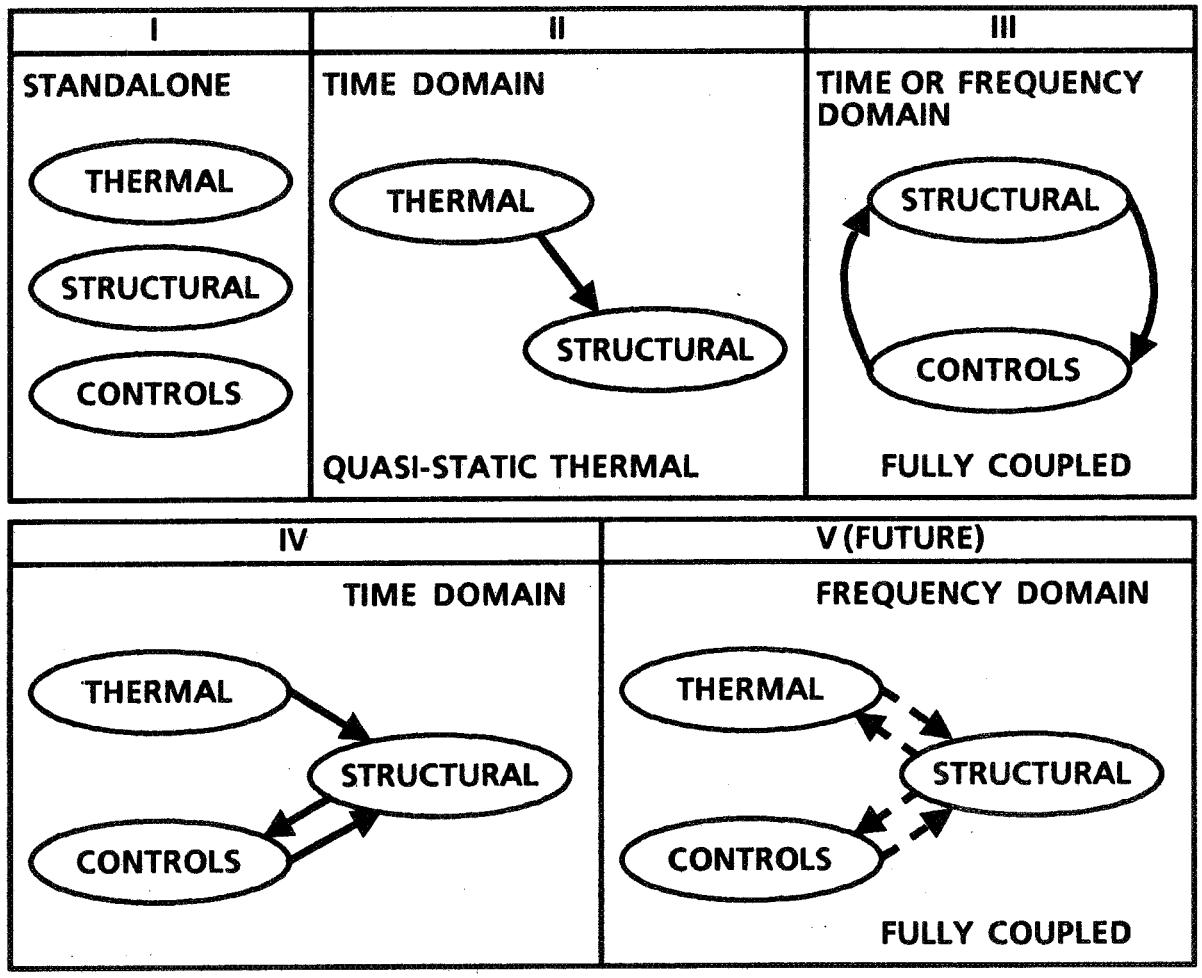


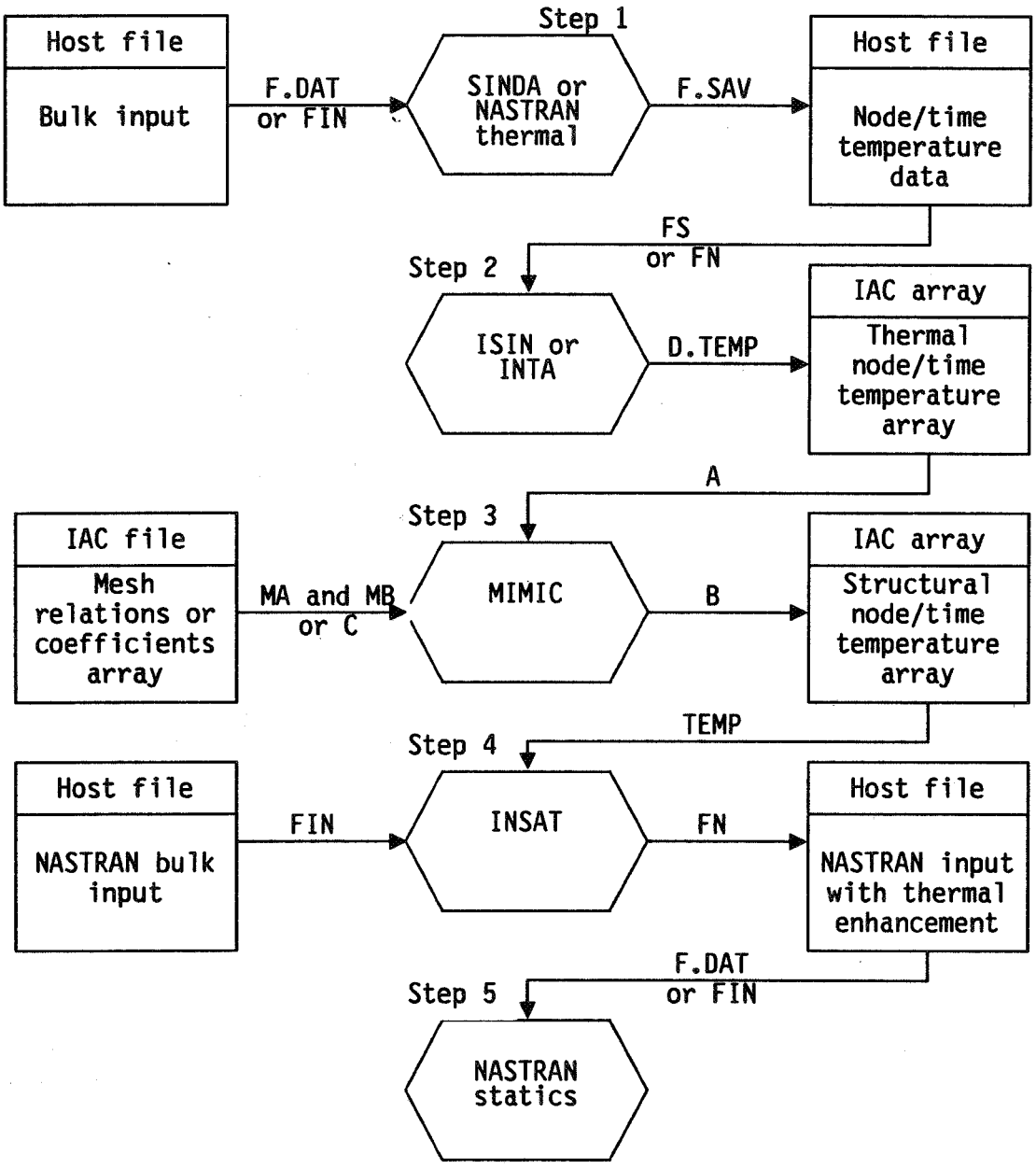
Figure 4.0-1: IAC Solution Paths

4.1 PATH I - STANDALONE

IAC Solution Path I is considered to be simply the set of solution capabilities provided by the individual technical modules, supplemented by the data management and graphics capabilities of the IAC system. Sections 2 and 3 of this manual have already discussed IAC capabilities from the viewpoint of the individual commands and modules which are available. Those sections are intended to be the primary reference for Solution Path I activities; they should also provide the user with insight into how the individual capabilities can be combined to form additional user-defined solution paths.

4.2 PATH II - THERMAL/STRUCTURAL

The IAC thermal/structural solution path uses an input thermal loading environment to compute steady-state or transient thermal deformations, via the coupling of a thermal analyzer and a structural statics analyzer. The current IAC Solution Path II is based on use of the SINDA or MSC NASTRAN module for thermal analysis, and MSC NASTRAN for the statics analysis. The flow diagram in Figure 4.2-1 shows the Solution Path II modules, data flow, and required RUN statements. Step 1 in this solution path is to run the thermal analyzer (either nonlinear steady state, or transient), in order to generate a file of node/time/temperature data. Step 2 is to run the ISIN or INTA module, which transforms these data into an IAC array. At this point the MIMIC module may be run in an optional step 3, in order to transform temperatures from the thermal mesh to the structural mesh. In step 4, the INSAT module is run. This extracts data at user-specified time points, transforms it into NASTRAN thermal-load-set form, merges these load sets and a time/subcase correlation table with a user-supplied NASTRAN statics input file, and generates an enhanced NASTRAN input file. Finally, step 5 runs the NASTRAN structural statics analyzer to compute displacements and stresses, including results for each thermal load case. (If desired, the statics analysis could be performed using NASTRAN Rigid-Format-Alter IACDMAP5, followed by execution of the INSA module, in order to store various computed quantities for query and evaluation.)

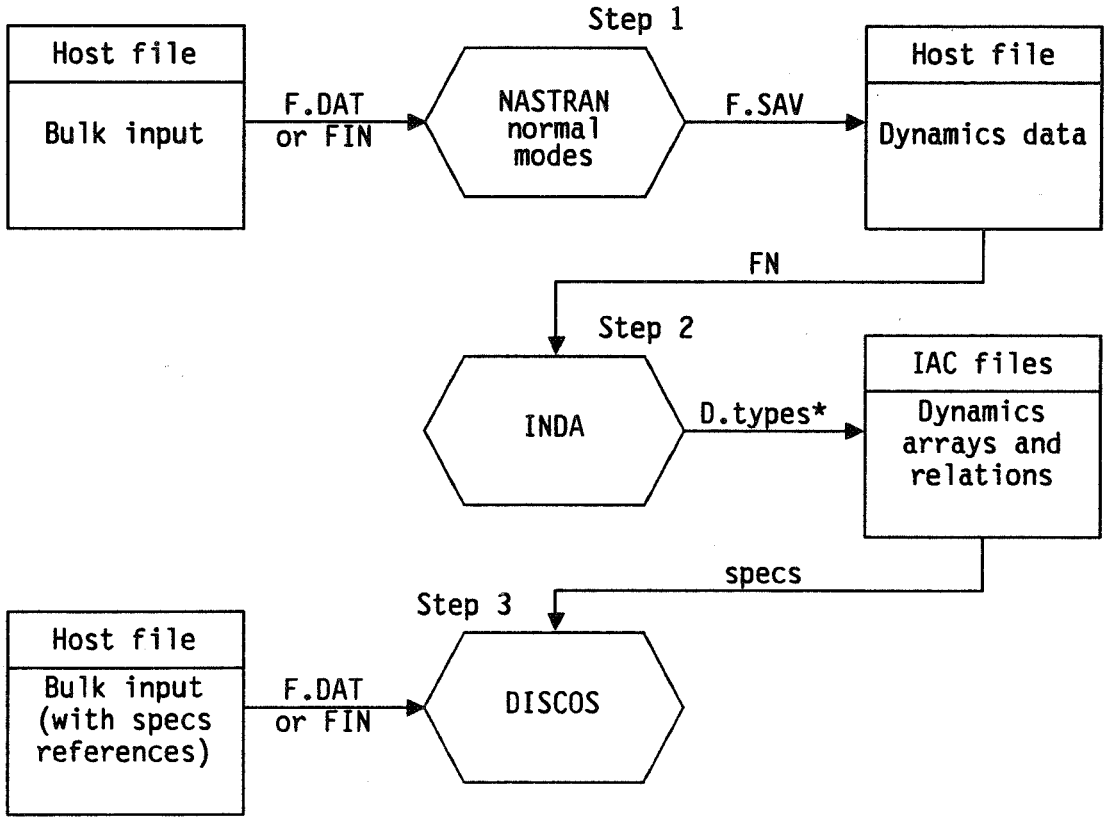



```
RUN SINDA (F=name, FIN=spec, INTERVAL=interval)
RUN NASTRAN (F=name, FIN=spec, RFA=IACDMAP2)
RUN ISIN (FS=spec, D=name)
RUN INTA (FN=spec, D=name)
RUN MIMIC (MA=spec, MB=spec, C=spec, A=spec, B=spec)
RUN INSAT (FIN=spec, TEMP=spec, TIME=(time-list), FN=spec)
RUN NASTRAN (F=name, FIN=spec)
```

Figure 4.2-1: Solution Path II - Thermal/Structural

4.3 PATH III - STRUCTURAL/CONTROL

The IAC structural/control solution path provides either a time-domain or frequency-domain analysis, via the coupling of a structural normal-modes analyzer and a system dynamics module. The current IAC Solution Path III is based on use of the MSC NASTRAN module for normal-modes analysis, and the DISCOS module for system dynamics (dynamics and controls) analysis. The flow diagram in Figure 4.3-1 shows the Solution Path III modules, data flow, and required RUN statements. Steps 1 and 2 in the solution path are performed for each flexible body in the system. Step 1 is to run a NASTRAN normal-modes analysis for the body. Step 2 is to run the INDA module. This module extracts dynamics data for user-specified mode numbers, contracts the data to eliminate any non-selected modes, and generates required IAC array and relation files. Step 3 is to run the DISCOS module, whose input file may contain IAC file references to dynamics data for all of the flexible bodies in the system, in order to perform either a time-domain or frequency-domain analysis. Results of the DISCOS system analysis can then be displayed and evaluated using the printer output and/or the interactive graphics capabilities.



* types = NODE, NMASS, MODE, MSTIF, MDAMP, MMASS

```
RUN NASTRAN (F=name, FIN=spec, RFA=IACDMAP1)
RUN INDA (FN=spec, D=name, MODE=(mode-id-list))
RUN DISCOS (F=name, FIN=spec)
```

Figure 4.3-1: Solution Path III - Structural/Control

4.4 PATH IV - THERMAL/STRUCTURAL/CONTROL

The IAC thermal/structural/control solution path provides a time-domain structural-control capability, as in Path III, but with the added capability to handle quasi-static time varying thermal loading effects. The methodology for Solution Path IV is described in Appendix F. It is assumed that thermal effects are weakly coupled with the system dynamics behavior, i.e. that the thermal loads can be computed a priori without direct regard for the dynamic changes in configuration. Although the primary purpose of this solution path is to handle quasi-static thermal loadings, the requirements for handling quasi-static mechanical loadings are virtually identical, and both types of loadings may be included independently or in a combined fashion.

IAC Solution Path IV is usually implemented using four technical modules - a transient thermal analyzer, a structural dynamic normal-modes analyzer, a structural static deformation analyzer, and a system dynamics analyzer. The current IAC solution path is based on use of the SINDA or MSC NASTRAN module for the thermal analysis, NASTRAN for normal-modes and statics analyses, and the DISCOS module for the time-domain system dynamics analysis. The flow diagram in Figure 4.4-1 shows the modules, data flow, and required RUN statements for operation of the last part of Solution Path IV. The diagram assumes that required data from the transient thermal and the normal-modes analyses are already existing as IAC files (see Solution Paths II and III), and therefore only the NASTRAN static deformation and DISCOS system dynamics modules are described.

Steps 1 through 4 in this solution path are performed for each flexible body in the system. Steps 1 and 2 involve the building of an enhanced bulk input file for the NASTRAN structural statics run, via the modules INSAT and INSAM; these two modules may be run in either order. INSAT uses an IAC node/time/temperature array to add thermal load sets, as described in Section 3.9. INSAM uses an IAC dynamic mode-shapes array to add appropriate scalar freedom and MPC definitions, as described in Section 3.10. The effect of the INSAM-generated data additions is to transform the NASTRAN static problem from a nodal (physical coordinates) basis to a modal (generalized coordinates) basis.

The mode-shapes definitions supplied to INSAM for the Solution Path IV analysis could be chosen by the user in any one of several forms:

1. Dynamics modes (e.g. created via NASTRAN IACDMAP1 and INDA) which are orthogonal or orthonormal with respect to the nodal mass or stiffness matrix.

2. Selected nonorthogonal thermal deformation shapes.

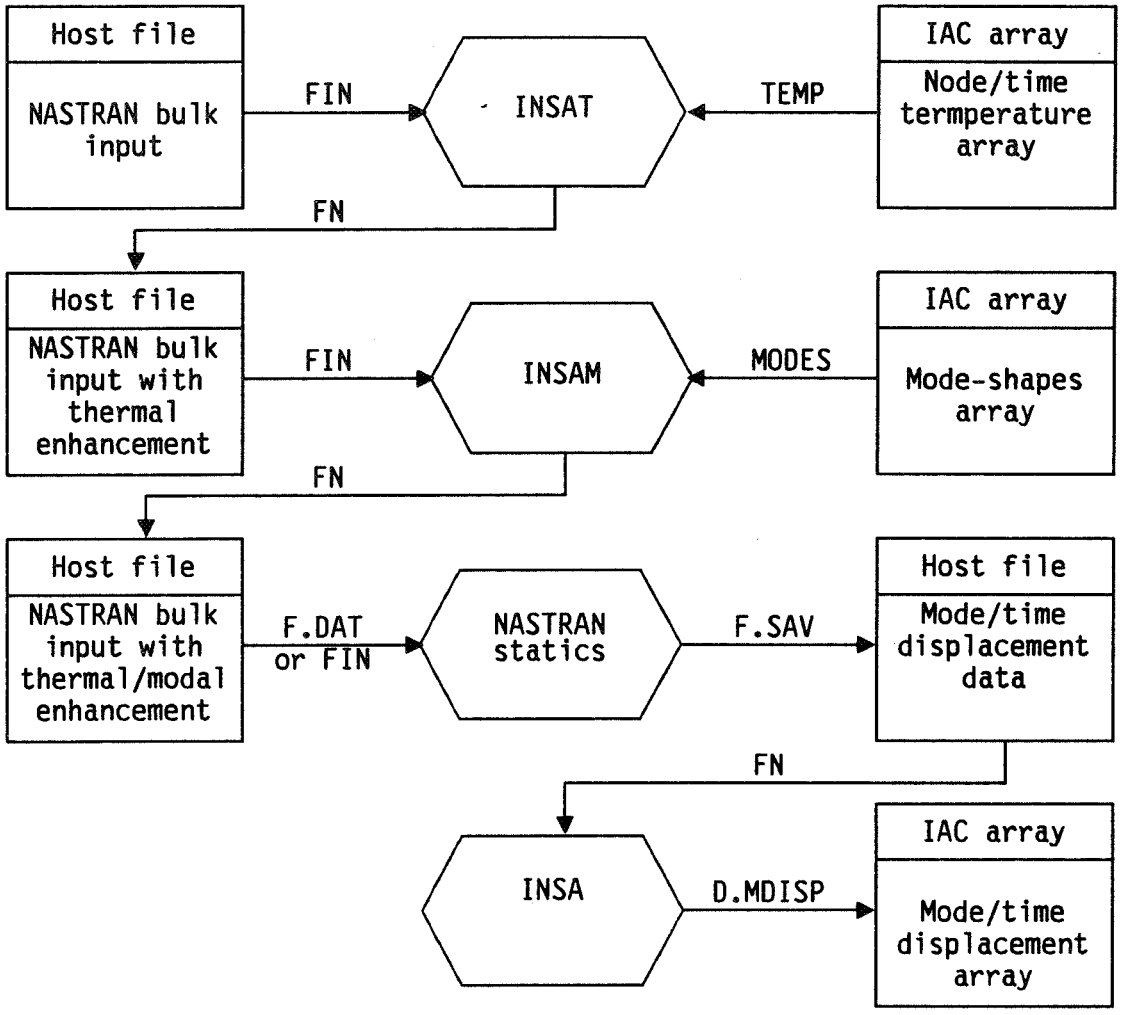
3. Arbitrary user defined mode shapes.

4. Some combination of 1, 2, and 3, above.

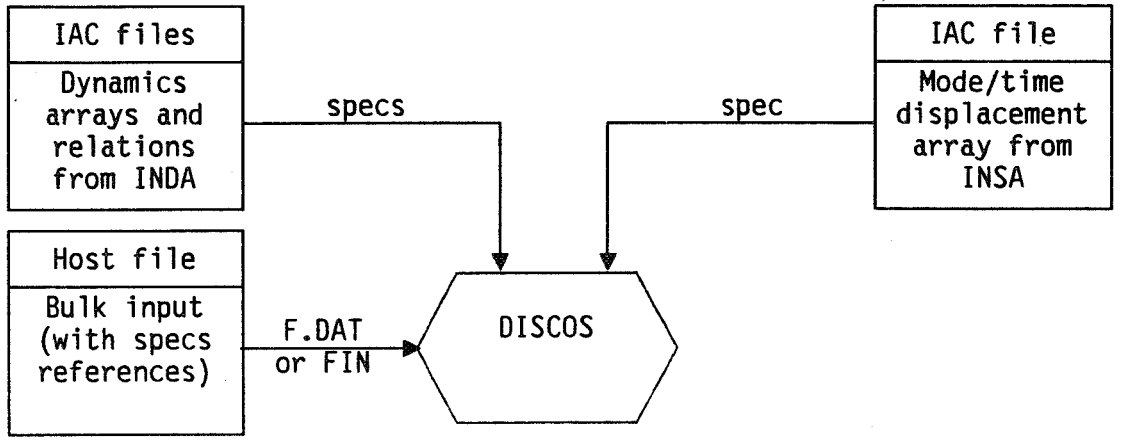
The selection criterion should be that the mode shapes must allow an acceptable simulation of the actual time history behavior, via a DISCOS time-domain analysis.

Step 3 is to run the NASTRAN statics module, in order to compute the output file of nodal and modal thermal displacement time histories. This is accomplished using the NASTRAN Rigid-Format-Alter sequence IACDMAP5, as discussed in Section 3.1.2. Step 4 is to run the INSA module, which inserts the required modal displacement array into the database.

Step 5 is to run a DISCOS time-domain analysis, including input quasi-static loads. These loads for Solution Path IV are in the form of a mode/time/thermal-displacement IAC array (one array per flexible body). DISCOS integrates the solution over time, using linear interpolation on the mode/time/displacement data to compute the quasi-static loads at each time step.




```
RUN INSAT (FIN=spec, TEMP=spec, TIME=(time-list), FN=spec)
RUN INSAM (FIN=spec, MODES=spec, FN=spec)
RUN NASTRAN (F=name, FIN=spec, RFA=IACDMAP5)
RUN INSA (FN=spec, D=name, TYPE=MDISP)
```




```
RUN DISCOS (F=name, FIN=spec)
```

Figure 4.4-1: Solution Path IV - Thermal/Structural/Control

THIS PAGE INTENTIONALLY LEFT BLANK

5.0 DATA ORGANIZATION AND STORAGE

The IAC approach to data organization and storage provides a formal database and data management system, complete with various types of data structures, detailed data manipulation and query, etc. The approach also provides for cataloging and the efficient storage and handling of less structured types of data, which are of critical concern to most computational modules. This section describes both of these very different but complementary IAC features.

The section is intended to be used for two purposes: 1) as a help for the more sophisticated users, who wish to understand the detailed structure and handling of IAC data; and 2) as a reference for application programmers, who need to call IAC utility routines and/or directly access and manipulate IAC data.

Section 5.1 describes the three types of IAC data areas (database, workspace and host file system). Section 5.2 summarizes the two types of IAC files (structured and unstructured). Sections 5.3 and 5.4 respectively describe the IAC structured data file types (RELATION, ARRAY and USER) in general, and the RELATION and ARRAY file types in particular. Finally, Section 5.5 discusses the three IAC provided techniques (integrated, interfaced and generic) for accomplishing data flow between different modules.

5.1 DATA AREAS

IAC provides for access to, and communication between, three types of data areas: 1) a concurrent multi-user database; 2) a RAM memory user-specific workspace; and 3) the ordinary host computer file system.

An IAC database is a permanent storage area containing a cataloged collection of IAC files. The workspace provides for direct user processing of IAC data (e.g. for data definition, manipulation and query) as well as for support of executive or application program data processing requirements. An IAC database or a host computer file may be accessible to many different users, while an IAC workspace is a private area accessible to only a single

user. IAC data transfer utilities (see Section 6.3) provide for various kinds of communication between the three types of data areas.

The following Sections 5.1.1, 5.1.2 and 5.1.3 deal with the respective database, workspace and host file system data areas.

5.1.1 DATABASE

The schematic of an IAC database is shown in Figure 5.1-1. The database is stored in, and is identified by the spec of, a host directory (or subdirectory). Only one IAC database may reside in a particular directory; conversely, the directory may contain additional data which is not associated with the database.

The physical database consists of a number of individual host files, two of which (the activities file and the catalog file) have special significance. The logical database consists of a number of 'IAC files' (cataloged files), each of which may consist of a data host file and/or an associated text host file. IAC files are discussed in Section 5.2. The remainder of this section is presented in two topics: the first describes the database activities, and the second describes the database catalog.

Database Activities - The database activities file is a sequential formatted file, used to control multi-user concurrent access to the database; the host spec of this file is IACACT.IAC. The activities file consists of 85-character records.

The first 4 records comprise an activities header, which describes the overall database characteristics.

Each succeeding record defines an activity, via the following format.

Item	Characters	Format
spec	1:43	A43
username	44:53	A10
code	54	A1
date	55:62	A8

time	63:73	A11
runid	74:81	A8
not used	82:85	---

Activities may be defined for three kinds of entities, as follows.

Entity	Spec	Code
overall database	$DATABASE	C,N
catalog	$CATALOG	R,W,L
IAC file	name:number.type;version	R,W,D,A,L

An overall database activity is added when the database is opened, and deleted when it is closed; the code defines the type of open access, and is either C (concurrent) or N (noconcurrent).

A catalog activity may occur when only the catalog attributes are to be processed, or when these attributes must be processed in order to process the cataloged IAC files; the code indicates the type of catalog processing, and is either R (read), W (write) or L (lock).

An IAC file activity code is either R (read), W (write), D (delete), A (archive) or L (lock). Note that the code A indicates either an archive or unarchive activity; L indicates a lock against all activities.

The following table defines the relationships between various activities on a particular entity.

Existing code	Codes allowed (other runid)	Codes allowed (same runid)
C	C	C
N	--	N
R	R	R
W	--	W
D	--	D
A	--	A
L	--	RWDAL

Database Catalog - The database catalog file is a sequential binary file, used to store the file level attributes for each IAC file in the database; the host spec of this file is IACCAT.IAC.

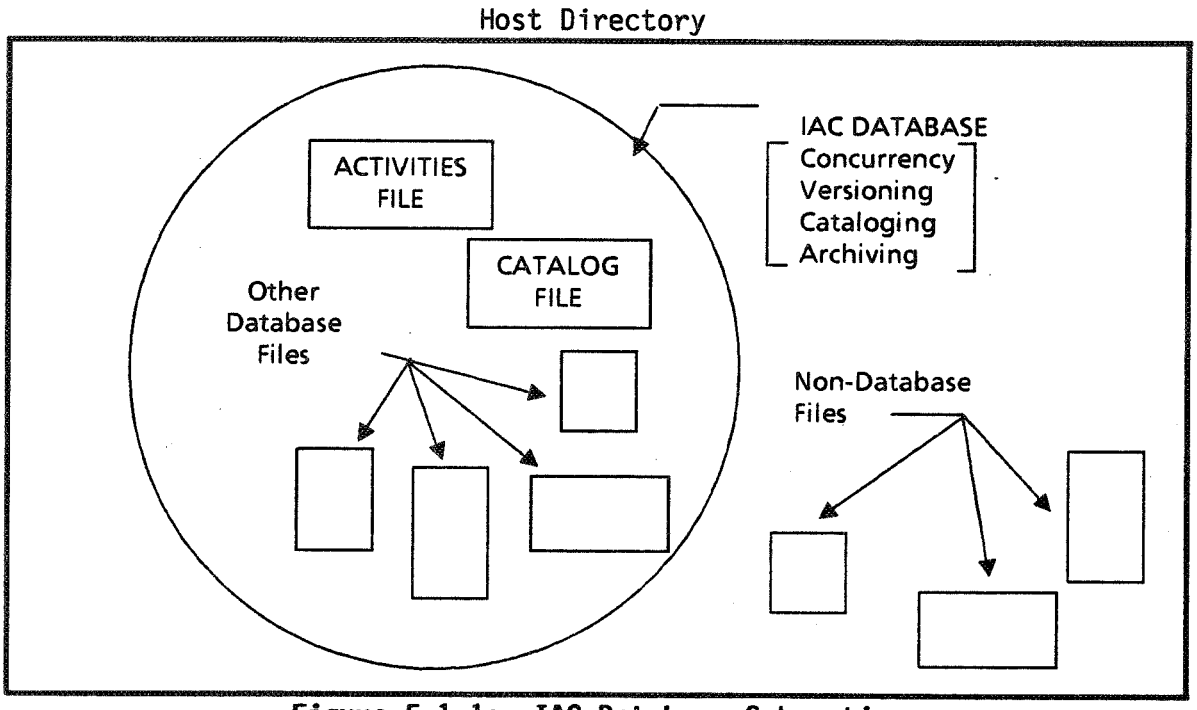


Figure 5.1-1: IAC Database Schematic

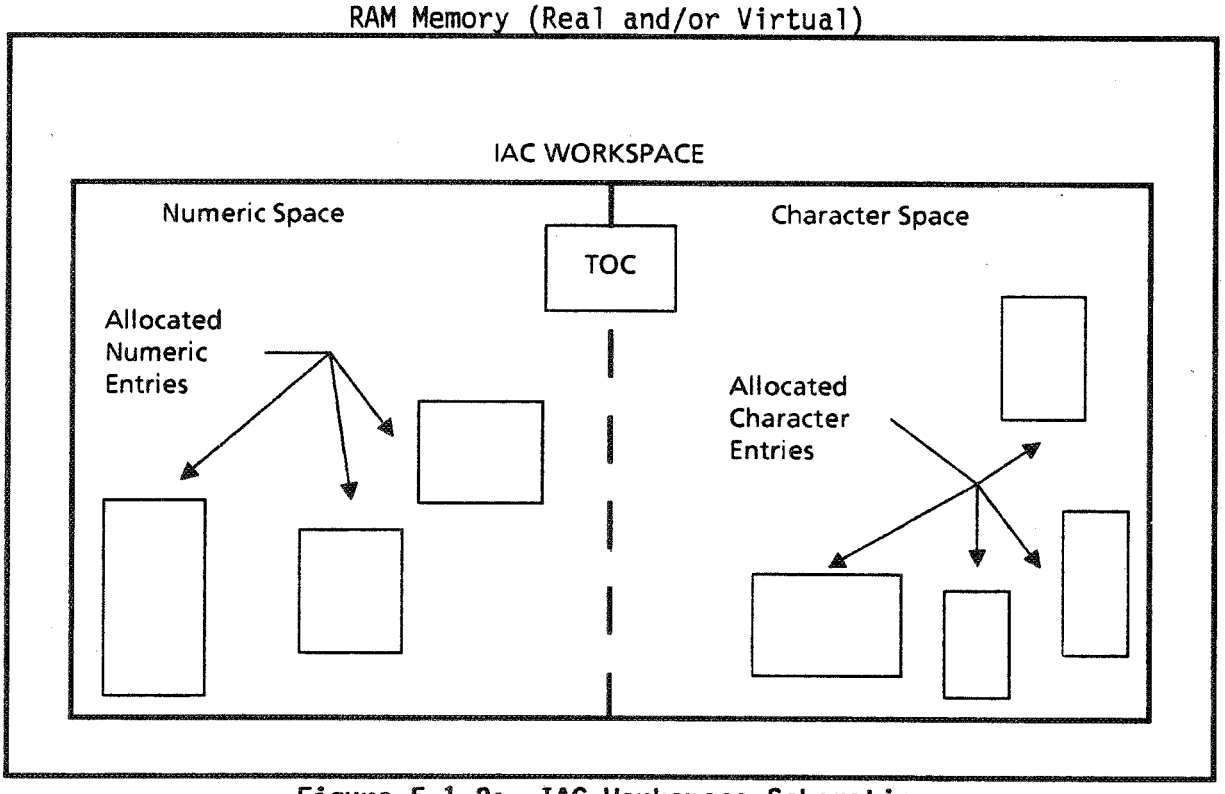


Figure 5.1-2: IAC Workspace Schematic

The database catalog file is a structured file of class RELATION (see Sections 5.3 and 5.4). It contains the following 16 attributes.

Name	Type	Format	Description
NAME	C10	1X,A10	IAC file name
NUMBER	I1	1X,I10	IAC file number
TYPE	C10	1X,A10	IAC file type
VERSION	I1	1X,I10	IAC file version
CLASS	C10	1X,A10	RELATION, ARRAY, USER, or ANY
STATUS	C1	1X,A1	N=normal, A=archived, I=incomplete
DATASPEC	C20	1X,A20	Data file host spec
TEXTSPEC	C20	1X,A20	Text file host spec
DATE	C8	1X,A8	Date cataloged
TIME	C11	1X,A11	Time cataloged
OWNER	C10	1X,A10	Owner (creator)
MODULE	C10	1X,A10	Creating module
PO	C5	1X,A5	Owner privileges (RWDAL)
PN	C5	1X,A5	Non-owner privileges (RWDAL)
KEYWORDS	C50	/1X,A50	Keywords identification string
TITLE	C80	/1X,A80	Title identification string

5.1.2 WORKSPACE

This section describes the architecture and application of the IAC workspace. The following topics describe the general approach, the workspace organization, and some specific information relative to the executive program workspace.

General Approach - Memory storage requirements within IAC tend to involve many different groups of data, and neither the number of such groups nor their individual sizes can be predetermined. The IAC approach to this problem is to assign, for each user, a RAM memory storage area (either real or virtual). Associated storage management software is provided to dynamically allocate the assigned memory into logical subareas as they are required. The host addressing and/or paging algorithm then supports the location and retrieval of data within the individual subareas, in a manner transparent to the IAC software. This memory storage area, and the associated IAC management software, is referred to as the IAC workspace.

The workspace was designed to support general application modules, as well as the particular needs of the executive program. A module may utilize an IAC workspace either as a simple dynamic buffer for data input and output, or in a more sophisticated manner to provide dynamic memory management for part or all of the module's data. The workspace provides memory operations such as allocation, deletion, extension and truncation. The workspace manager software is provided as a set of documented utility subroutines (see Section 6) in order to facilitate its utilization by application module programmers.

Workspace Organization - The workspace is an area of real or virtual memory, unique to each IAC user and module. The workspace consists of two FORTRAN 77 array areas: one dimensioned for 'lnum' numeric storage units and the other dimensioned for 'lchr' character storage units. Each area is dynamically broken into subareas called entries, which are the basic elements targeted by the workspace utilities. See also Section 6.1.

An entry can be a vector from an IAC table or array, in which case its data elements are all of the same type. However, this is not required; for example, each record in an IAC USER class file may become an entry in an IAC workspace.

All entries are described in the workspace TOC (table of contents). The TOC is a table whose rows refer to entries in the workspace and whose columns contain identification, size and location information. As entries are added, modified and deleted, the workspace manager updates the TOC information.

The TOC contains seven numeric-type columns and two character-type columns, as described in Figure 5.1-3a.

When a module requests the creation of a workspace, it specifies the size of the numeric and character areas (see utility IACBEG in Section 6). Four active rows are initially defined in the new TOC. The first two rows are used by the TOC for its own self description, since the TOC storage itself requires a numeric and a character entry. The next two rows describe the initial free space. The initialized TOC configuration is shown in Figure 5.1-3b. There the number of rows in the TOC is denoted by 'n', the size in

EID	ELEN	ELOC	ESOB	ENOB	NUL1	NUL2	ENAM	ETYP

EID (Entry ID) This is an integer which is the basis for communication between the workspace user and manager. Its values are:
+ for numeric entries (EID = row number in TOC)
- for character entries (EID = - row number in TOC)
0 for unassigned entries (inactive, unused TOC rows)

ELEN (Entry LENgth) This integer is the size in number of FORTRAN numeric or character 'units' of the entry. Special values are:
+ for allocated entries
- for free space entries

ELOC (Entry LOCation) The index of the first storage location of the entry in numeric or character units is in this integer.

ESOB (Entry Size Of Block) Entries are stored by physical subunits called 'blocks' which are constant in size (measured in number of numeric or character units). The size of these blocks is given in this integer. Special values are:
+ permits the workspace manager to increase or decrease the ENOB
- permits the manager to increase but not decrease the ENOB

ENOB (Entry Number Of Blocks) This integer is the number of blocks assigned to the entry.

NUL1, NUL2 (NULL spaces 1 & 2) These numeric spaces are not normally used. However, the workspace manager may use these columns for sorting and other randomly occurring operations.

ENAM (Entry NAMe) This 10-character string is the name of the entry. Special system names begin with a "$".

ETYP (Entry TYPe) This 10-character string is the type of the entry. Special system types begin with a "$".

(a) TOC Column Definitions

EID	ELEN	ELOC	ESOB	ENOB	NUL1	NUL2	ENAM	ETYP
1	7n	1	1	7n	--	--	TOC	NUMERIC
-2	20n	1	1	20n	--	--	TOC	CHARACTER
3	-(lnum-7n)	7n+1	1	lnum-7n	--	--	FREESPACE	NUMERIC
-4	-(lchr-20n)	20n+1	1	lchr-20n	--	--	FREESPACE	CHARACTER
0	--	--	--	--	--	--		
	etc.							

(b) TOC Initial Configuration

Figure 5.1-3: Workspace TOC

numeric units of the numeric area is 'lnum', and the size in character units of the character area is 'lchr'. Rows 5 through 'n' are initially marked as unassigned, with EID = 0. The TOC can be increased or decreased in size as required. The workspace manager keeps track of frequently needed items of information about the TOC such as 'n' or the currently available number of unassigned rows.

The workspace manager utilities are defined in detail in Section 6. Users should always go through an IAC utility to modify or access the workspace TOC, or the TOC may become corrupted.

Executive Program Workspace - The ACE executive program utilizes an IAC workspace to support most of its operations, including command processing, module execution, and data manipulation and query. By utilizing the workspace concept, the executive has been able to avoid virtually all of the arbitrary and annoying potential restrictions on individual sizes (for example, the maximum number of conditions in a user query, the maximum size of an input command or an output logical data group, or the maximum number of parts in a data structure).

A major use of the workspace within the executive program involves the local storage and manipulation of IAC files. In order to manage these files, the executive maintains a workspace catalog whose organization is somewhat similar to that of the database catalog. The executive workspace catalog may be considered to logically contain the following attributes.

Name	Type	Description
SPEC	C43	IAC file spec (name:number.type;version)
CLASS	C10	RELATION, ARRAY, USER or ANY
DATAPSEC	C20	Data file host spec
TEXTSPEC	C20	Text file host spec
PO	C5	Owner privileges (RWDAL)
PN	C5	Non-owner privileges (RWDAL)
KEYWORDS	C50	Keywords identification string
TITLE	C80	Title identification string
EIDEID	I1	Entry ID list

The EIDEID attribute is the ID of the workspace entry which contains the list of all entry ID's for the data file. EIDEID provides access to these entries; it is used by the executive to display information such as the number of entries and the total logical sizes (numeric and character storage units) of all combined entries in the file.

5.1.3 HOST FILE SYSTEM

The host file system may be used within IAC for several different purposes. Typical functions of the system are briefly summarized in the following topics.

Support of the Database - As indicated by Figure 5.1-1, the IAC database architecture extensively utilizes the host file system capabilities. The database activities, the database catalog, and the database cataloged data and text information (see Section 5.2) are all organized as individual host files in the database directory.

This form of organization allows a user, when necessary, to access information in the files directly via host operating system commands. It is an efficient approach for storing arbitrary application program formatted data (unstructured files) in the database, because it does not require copying and reformatting of that data. Finally, it tends to increase the robustness of the database, because operating anomalies (e.g. hardware or power failures, software errors) do not destroy the integrity of the database through broken addressing chains, etc.; any resulting data destruction is localized to a single file, and standard host operating system tools can be used for backup and recovery.

Logical Extension of the Workspace - The host file system is also used as a logical extension of the IAC workspace, during storing and processing of data and text files. For example, the IAC basic file transfer utilities (IACGET and IACPUT) serve to transfer files between the database and the 'workspace'; the actual workspace is used for structured data files, while the user's host file system directory is used for text and unstructured data files. This

provides for 'native mode' editing and manipulation of the text and unstructured files, using standard operating system capabilities.

Support of Arbitrary User Functions - The host file system supports several IAC related user functions. The ACE HOST command allows a user, from within the ACE executive program, to execute any sequence of host operating system commands; these may include various operations involving the host file system. Executive program input commands or data, and various types of executive output, may be directly supported by the host file system.

The executive generic data-flow capability (see Section 5.5) provides a very important interface to the host file system. By this means, selected data within character formatted host files can be communicated to or from structured data files in the executive workspace; this data can in turn be manipulated and queried, or transmitted to or from an IAC database.

An IAC structured file (RELATION, ARRAY or USER class) may consist of a data and/or a text host file. These host files may, if necessary, be individually accessed by the user.

Finally, an IAC unstructured file (ANY class) is a host file, in its originaly created form, which has been cataloged into an IAC database. Application modules often logically treat such files as host files, and read them directly.

5.2 IAC FILES

This section gives the definition of an IAC file, explains the syntax of an IAC file spec, and summarizes the IAC file organization.

IAC File Definition - Logically, an IAC file is a group of information, which is cataloged in an IAC database or in an ACE executive workspace. Physically, the IAC file consists of a data file, which contains 'structured' or 'unstructured' data (see last topic in this section), and/or a text host file, which contains textual information describing the contents of the IAC file. The database or workspace catalog contains associated information, such as title, keywords, class and access privileges.

IAC File Spec - An IAC file spec has the four-part form

name:number.type;version

where ":number" and ";version" are optional user inputs.

The name and type are maximum 10-character alphanumerics, with first character alpha. Alpha includes upper case A-Z and underscore "_". The number and version are any integers representable via the FORTRAN "I10" format. If the number is not specified, the default is 1. If the version is not specified, the default is the highest version for an existing spec, or the maximum of (1,highest+1) for a new spec.

A reference to an existing IAC file spec may in some contexts (e.g. in the IACUNC utility or the executive GET command) include wild cards "*" within any part of the spec, to indicate "any existing", i.e. any null or non-null sequence of characters. A definition for a new IAC file spec may in some contexts (e.g. in the IACREN utility or the executive RENAME command) include a single wild card "*" for any part of the spec, to indicate "same as existing part".

IAC File Organization - As noted previously, an IAC file physically consists of a data file and/or a text host file. The data file may be stored,

depending upon the context, either as a host file or as a number of entries in the workspace. Use of an IAC file spec generally indicates an operation on the entire logical IAC file; for example, the IACDEL utility or the executive DELETE command deletes both the data file and the text host file.

The text host file contains user defined textual information describing the contents of the IAC file. The text file is character formatted.

The data file may be 'unstructured' (IAC class is ANY). In this case the file must exist as a host file, either in the database host directory if the IAC file is logically in the database, or in the user host directory if the IAC file is logically in the workspace. The IAC system does not alter, or insert any information into, the data file during the cataloging process.

Alternatively, the data file may be 'structured' (IAC class is RELATION, ARRAY or USER). In this case the file may exist in one of two ways, either as a host file in the database if the IAC file is logically in the database, or as a number of entries in the workspace if the IAC file is logically in the workspace. The detailed organization of IAC structured data files is described in the next Section 5.3.

5.3 STRUCTURED DATA FILES

A structured data file contains data in an IAC standard organization and format. The logical IAC file, of which the data file is a part, is designated as belonging to one of the classes RELATION, ARRAY or USER. Generation of data in the form of a structured file will allow various kinds of transfer of the data via IAC standard utilities; the RELATION and ARRAY classes of files also enable detailed query of their data via ACE executive commands.

5.3.1 GENERAL CHARACTERISTICS AND APPLICATION

Figure 5.3-1 illustrates typical forms for the ARRAY, RELATION and USER class data structures (data files). Each class can be most effective for particular applications.

The ARRAY is probably the most important IAC data structure. It is effective for both computational and query purposes, and is used to support most of the IAC technical module interfaces. This structure is a generalization of vector and matrix type data, and can be considered to include the RELATION structure as a subset. The ARRAY shown in Figure 5.3-1 is 2-dimensional and contains transient nodal temperature data. The core of the general structure consists of one or more arrays of arbitrary order (e.g. a matrix may be represented as a second order ARRAY), and each dimension (index) may be of arbitrary size. In order to facilitate user query of the data, each index may have an associated table (relation) consisting of one or more labels. Each part of the structure (i.e label or array) can be referenced by name. An important feature of the ARRAY is the ability to also reference data by index (IAC provides a linear array index as well as an index for each of the dimensions). The ARRAY data structure is used to handle much of the data for IAC modules and solution paths, e.g. nodal temperature data, mode shapes, mass and stiffness matrices, displacement and stress data, and plant/controller definition matrices.

The RELATION data structure represents table type data. It may be of arbitrary size, as measured by its number of columns and number of rows.

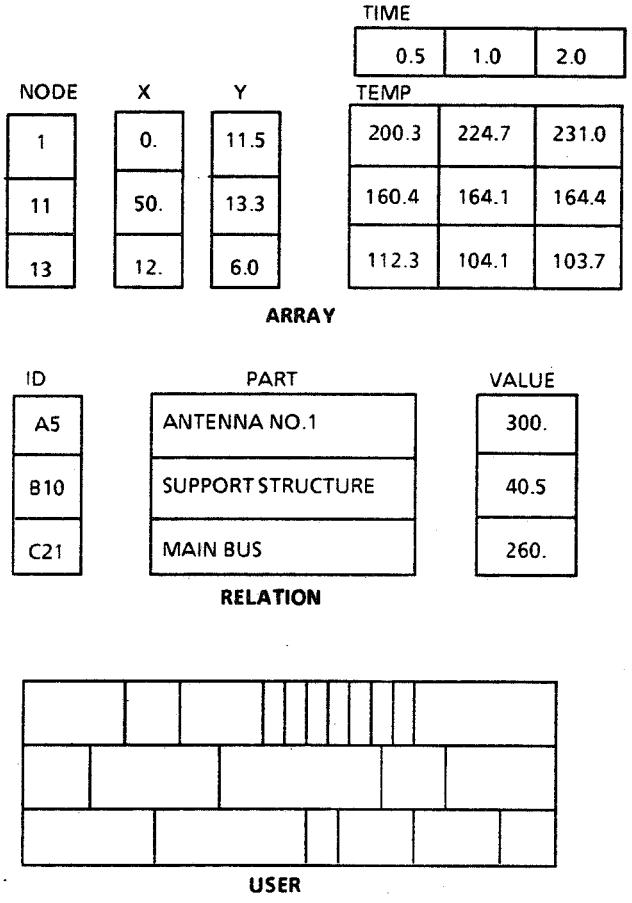


Figure 5.3-1: IAC Data Structures

Each column can be referenced by name, and values may also be referenced by index (row number). The RELATION shown in Figure 5.3-1 contains simple configuration type data. For many applications the RELATION is inferior to the ARRAY, in terms of storage and computational efficiency, as well as query capabilities. However, IAC usage has shown the RELATION to be effective in representing management type data (e.g. data on costs, schedules and personnel), in defining project configurations, and for numerically representing certain types of graphics data. The IAC database catalog is another special type of relation structure. The relational form also can be useful in communications between different integrated systems, since other data forms can usually be converted into logically equivalent relational form.

The USER data structure provides a user with considerable flexibility in defining arbitrary groups of data and relationships among them. IAC facilities handling of USER data by providing a standard means of organizing the user defined data groups, and of managing the associated data types, data group identifiers, and sizes.

5.3.2 PHYSICAL ORGANIZATION

A structured data file is stored in the host file system as a binary sequential file. (It is generally accepted in the data management field that if more than 5 percent of a data set is to be processed, then sequential access is most efficient; this is the case for most engineering applications.) There are three types of records in the file; these records are referred to, respectively, as Record 0, Record 1, and Entry Records. Each type of record is described in one of the following topics.

Record 0 - Record 0 consists of a single 30-character string, defined as follows.

Item	Characters	Format	Description
IAC	1:10	A10	IAC identifier
class	11:20	A10	RELATION, ARRAY or USER
unit	21	A1	Record 1 unit flag (C)
size	22:30	I9	Record 1 size

The data is binary; however, the Format column indicates how the data could be transferred (via reads or writes) between an internal character array and appropriate FORTRAN variables. (Note that some machines do not even differentiate between binary character and formatted character data.) In other words, the character values in the record are left justified, and the integer value is right justified.

The unit and size values enable reading of the following Record 1. The unit value is always C, indicating that Record 1 contains character data; the size value indicates the number of character storage units contained in Record 1.

Record 1 - The next record, referred to as Record 1, is an ordered list of 30-character strings, each of which describes a succeeding record (an Entry Record) in the file. The size of Record 1, in characters, is the size value from Record 0; it is equal to 30 times the number of Entry Records. Each 30-character string is defined as follows.

Item	Characters	Format	Description
name	1:10	A10	Entry (record) name
type	11:20	A10	Entry type
unit	21	A1	Entry unit flag (N or C)
size	22:30	I9	Entry size

The Format column is to be interpreted in the same manner as for Record 0.

The unit and size values enable reading of the corresponding Entry Records. A unit value of N indicates that the entry (record) contains numeric data, and C indicates character data; the size value indicates the number of numeric or character storage units contained in the entry.

The name and type together comprise the name.type identifier for the entry. Each name or type is a 10-character alphanumeric, with first character alpha. Alpha here includes upper-case A-Z, underscore "_", and the system reserved designator "$". Each entry identifier within a particular file should be unique.

The following table provides a list of currently defined entry names and types, and the corresponding structured file classes (A, R, U respectively denote ARRAY, RELATION, USER) in which the entry may be contained.

Name	Type	Classes	Description
name	type	A,R,U	General user defined vector
$RELATION	$DES	R	Relation descriptor
$ARRAY	$DESO	A	Array index 0 descriptor
$ARRAY	$DESi	A	Array index i descriptor
name	$ATT	A,R	Attribute values
name	$ATTL	A,R	Attribute element lengths (numeric or character storage units)
name	$ATTi	A,R	Nonscalar attribute index i size (elements)
name	$RULE	A,R	Data validation rule (conditions)

Any of the three classes of structured files may contain general user defined vector entries. The name and type for such an entry are arbitrary, as long as they follow the convention defined above, and do not contain "$". The general user defined entry may contain an arbitrary group (vector) of numeric or character data. The entry may be used within a USER class file as part of a user defined data structure; it may also be stored within a RELATION or ARRAY class file, as additional information not associated with the standard defined data structure.

The types beginning with $DES are used to store descriptor information for a RELATION or ARRAY (see Section 5.4).

The types beginning with $ATT contain information for a particular attribute in a RELATION or ARRAY; the attribute is identified by the associated name.

The $ATT (with no catenated characters) type entry is always present for each attribute and contains the attribute values. The other $ATT (with catenated characters) type entries may be used to define additional information needed for the attribute. In order to explain these other types, let the following parameters be defined.

nt = number of tuples (rows in a RELATION, or in an ARRAY associated relation)

ne = number of stored data elements in an attribute; for a scalar attribute, ne = nt; for a nonscalar attribute, ne = sum (over nt) of products of the attribute index sizes.

nu = number of storage units (numeric or character) used per data element.

The $ATTL entry, if present, indicates variable length elements; it is currently used only for character attributes defined with a C* data type. Using the above parameter definitions, the $ATTL entry contains ne integer lengths (values of nu), and gives the number of storage units in each data element, i.e. the number which may be thought of as replacing the variable length * indicator.

A $ATTi entry, if present, indicates variable size associated with index number i of a nonscalar attribute. Each $ATTi entry contains nt integer sizes. For example, an attribute having a subscript definition [*,5,*] would have associated entry types $ATT1 and $ATT3.

The $RULE entry type contains a standard IAC condition, in character string form, for checking the validity of data within a RELATION or ARRAY (e.g. after loading or changing). The data is valid if a TALLY type query results in selection of all tuples from the file.

5.4 RELATION AND ARRAY FILES

Section 5.3 describes the general characteristics and physical organization of IAC data structures (structured files). The present section provides some additional details of the physical organization for RELATION and ARRAY files, in particular the organization of their descriptor entries.

Entry types beginning with $DES are used to store descriptor information for a RELATION or ARRAY. This information defines how the other entries are logically related within the overall data structure.

Each descriptor is stored as a character entry (record). A RELATION has one descriptor. An ARRAY might logically be considered to be a group of one or more associated relations; the ARRAY contains at most one RELATION-like descriptor entry for each index (for the linear index 0, and for each of the dimensioning indexes 1, 2, etc.). As an example, consider a 2-dimensional ARRAY, with an associated table (relation) corresponding to index 2. This data structure would have a $ARRAY.$DES0 entry describing the characteristics of the core array, and a $ARRAY.$DES2 entry describing the characteristics of the associated table.

Each descriptor contains an initial string of 50 characters, which describes overall characteristics of the relation (or core array). This is followed by another string of 50 characters for each attribute. Note that the core of an ARRAY structure may contain more than one attribute, i.e. multiple arrays of possibly different data types may be logically stacked together.

The items defined within the initial 50-character string, and each succeeding 50-character string, are described below.

Item	Characters	Format	Description
na	1:5	I5	Number of attributes
nt	6:15	I10	Number of tuples
ardim	16:35	A20	Array dimensions (blank for relation)
--	36:50	--	Not used

name	1:10	A10	Attribute name
atdim	11:20	A10	Attribute dimensions (blank for scalar)
type	21:25	A5	Attribute data type
format	26:50	A25	Output display format

The Format column is to be interpreted merely as a method for reading values from, or writing them to, an internal character array (see Section 5.3.1, Record 0 topic).

If the descriptor is associated with a linear array index 0, the item na often has the value 1; however, it may be larger if the file contains multiple arrays. The size nt is the size of the associated index. The item ardim is blank except for an array linear index 0; in that case it defines the array dimensions, with the product of the dimensions equal to nt; ardim contains no internal blanks, and its dimensions are separated by commas.

The item name is an alphanumeric (see Section 5.3).

The item atdim has a value similar to ardim, and defines the index sizes (dimensions) for a nonscalar attribute; each size is either a positive integer value, or the value "*" which indicates variable size.

The item type has one of the values I1, R1, R2, Z2, Z4, L1, Cn or C*; the n is an integer value, and "*" denotes variable length character string.

The item format may be any FORTRAN 77 compatible format (outer parentheses not required), valid for output display of the scalar attribute or nonscalar attribute element.

The following requirements should be noted for ARRAY and RELATION files.

1. A RELATION or ARRAY may not have a descriptor which defines no attributes.

2. Array dimensions must be greater than zero.

3. The descriptor for a core array (index 0) must exist and must be consistent with any other existing descriptors (index 1, 2, etc.) for the ARRAY.

4. The descriptor for a core array (index 0) may not contain "*", i.e. all nonscalars must have fixed dimensions and data types must be fixed length.

5. All entry identifiers (name and type) must be unique within the file.

6. A file may contain entries other than those referenced by a descriptor.

5.5 DATA FLOW TECHNIQUES

IAC facilitates the flow of data between different modules or between a module and the user, by providing a central database storage area, standard data structures and formats, and various data management tools. In order to serve the needs of diverse modules and users for communicating with the database, IAC provides three different data flow techniques. Referring to Figure 5.5-1, these techniques are denoted respectively by the terms "integrated," "interfaced," and "generic." Each technique is useful for certain types of applications depending upon the particular module characteristics, the resources available, and current and future user needs. In describing these techniques here the emphasis is on dealing with already existing modules, but many of the same considerations apply even if a new module is being developed specifically for use within the IAC system.

In the integrated approach, IAC database read/write utilities are linked directly into the module, along with any necessary code for conversion of data structures and formats. This approach can be effective if a fairly mature module source code is available, programming personnel are familiar with the module code, and the current and future data flow requirements are well defined. Integrated data flow has been provided for IAC modules such as DISCOS and ORACLS.

The interfaced technique uses an intermediate module, called an interface module, to bridge between the database and a module formatted binary file. (In some cases, especially for database-to-module flow, a character formatted file such as a module card image input file may be utilized.) This technique can be used effectively where module source code is not available or the software is difficult to modify, and the module is capable of reading or writing the required data via a module defined file. It is also helpful where users wish to save significant quantities of module computed data on an output file, and then to decide at some later time which selected items of data are to be converted and stored in the database. Interface modules are used to provide data flow for IAC modules such as SINDA and NASTRAN. In the case of NASTRAN, several different interface modules have been developed; some of these modules process NASTRAN generated binary files to extract

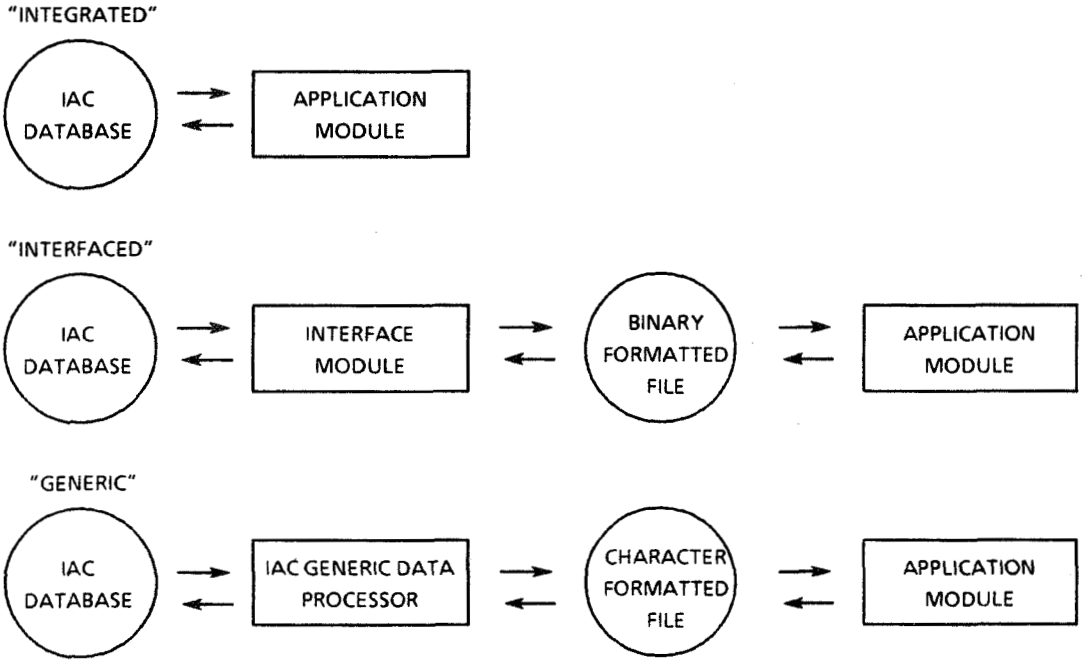


Figure 5.5-1: IAC Data Flow Techniques

computed nodal temperatures, static displacements, normal modes data, etc.; other modules use data in an IAC database to generate enhanced bulk input files for NASTRAN containing added data such as thermal load sets or constraint equations.

The generic data flow aproach is characterized by a human-readable character formatted file, and user-in-the-loop control. The process tends to be less automated than for the integrated or interfaced technique, but provides considerable flexibility and may in some cases prove to be more efficient. As implemented within the IAC executive, the generic data processor provides the user with sophisticated tools for selecting and formatting the data to be read or written. Data may be selected based on criteria such as line positions or groups, character positions or values, presence or absence of certain character sequences, or similar criteria applied to free-field data items within the line. The IAC generic data flow capability is often used to extract data from printed output files, e.g. NASTRAN stress data, so that the data can be queried and plotted. It is helpful in generating partial input files, e.g. a list of element or node point definitions from data available in the database. The generic technique has many other uses as well, such as in transferring data between two different types of host computers.

6.0 IAC UTILITIES

The IAC Utilities are procedures that are callable by user programs as well as by IAC application modules and the ACE Executive.

All IAC utility procedures and common blocks are named with 'IAC' as the first three characters. Programmers are encouraged NOT to name their own procedures and common blocks starting with these letters so that utility references can be easily identified.

IAC utility software is classified into two usage groups, identified by the respective procedure naming conventions IACxyz and IACdxy. Here "x", "y" and "z" denote letters, and "d" denotes a digit. The IACxyz utilities are intended for general use; they are documented in this section, are usually fully FORTRAN 77 compliant, and a committment has been made to their future upward compatibility. The IAXdxy procedures may be called by the IACxyz utilities, but are not intended for direct use by application programmers; they are not documented in this section, and are subject to upward incompatible changes.

IACdxy Procedures - The following conventions describe the "d" digit.

Digit	Description
0	programmed in assembler language.
1-2	Not assigned
3	Programmed in FORTRAN, but are dependent upon machine job control language, system routines and/or special features.
4	Programmed in extended FORTRAN 77, e.g. use double precision complex variables
5-7	FORTRAN 77 compliant
8-9	Not assigned.

IACxyz Utilities - The set of IACxyz utilities is expected to continue to expand, as the IAC system develops and additional programmer requirements are identified. Documentation on these utilities is summarized by the following paragraphs.

Section 6.1 describes the Workspace Utilities, which aid in managing the RAM memory area assigned to each user. These utilities are named 'IACWxx' to identify them. They can be used to find, describe, allocate and deallocate 'entries' in the Workspace.

Section 6.2 is concerned with the Database/File Management Utilities; these can be used to manage the database, or to manipulate whole files without the necessity of copying them into the workspace first. However, editing or searching type functions here deal only with the database catalog, and not with individual files.

Section 6.3 describes the File Transfer Utilities. These move files or portions thereof between the IAC Database, the Workspace, and Host Files.

Section 6.4 describes Utility Condition Processing. Information is given concerning condition processing variables and the message utility.

Miscellaneous Utilities are described in Section 6.5. The general purpose functions available to the programmer but not described in a previous section are given here.

Last is Section 6.6 consisting of an abbreviated summary of all utilities.

6.1 WORKSPACE UTILITIES

A Brief Review of the Workspace - The Workspace is an area of RAM memory unique to each IAC user. In it occur all data manipulations not treating an entire file as an amorphous entity. The Workspace consists of two FORTRAN 77 array areas: one dimensioned for 'lnum' numeric storage units (which includes logical data) and the other dimensioned for 'lchr' character storage units. Each area is dynamically broken into subareas called entries, which are the basic elements targeted by the Workspace utilities. The two respective areas are defined by the following user accessible common blocks.

```
COMMON /IACWAN/NWORK(lnum)
COMMON /IACWAC/CWORK(lchr)
```

An entry may be described simply as a group of consecutive numeric or character storage units. An entry can be a vector from a table or array, in which case its data elements are all of the same type. However, this is not required; for example, a record in a USER class file corresponds to an entry, and may contain an arbitrary collection of data value types.

All entries are described in the Workspace Table of Contents (abbreviated to TOC). The TOC is a table whose rows refer to entries in the Workspace and whose columns contain identification, size and location information. As entries are added, modified, and deleted, the Workspace Manager updates the TOC information. The following topic subsection describes the TOC more fully.

Workspace TOC Description - The Workspace Table of Contents (called the TOC) is a nine column table with a variable number of rows, each row corresponding to a Workspace 'entry'. There are seven numeric-type columns and two columns of character type, as described below.

EID	ELEN	ELOC	ESOB	ENOB	NUL1	NUL2	ENAM	ETYP

EID (Entry ID) This is an integer which is the basis for communication between the Workspace user and manager. Its values are:
+ for numeric entries (EID = row number in TOC)
- for character entries (EID = - row number in TOC)
0 for unassigned entries (inactive, unused TOC rows)

ELEN (Entry LENgth) This integer is the size in number of FORTRAN numeric or character 'units' of the entry. Special values are:
+ for allocated entries
- for free space entries

ELOC (Entry LOCation) The index of the first storage location of the entry in numeric or character units is in this integer.

ESOB (Entry Size Of Block) Entries are stored by physical subunits called 'blocks' which are constant in size (measured in number of numeric or character units). The size of these blocks is given in this integer. Special values are:
+ permits the Workspace manager to increase or decrease the ENOB
- permits the manager to increase but not decrease the ENOB

ENOB (Entry Number Of Blocks) This integer is the number of blocks assigned to the entry.

NUL1, NUL2 (NULL spaces 1 & 2) These numeric spaces are not normally used. However, the Workspace Manager may use these columns for sorting and other randomly occurring operations.

ENAM (Entry NAMe) This 10-character string is the name of the entry. Special system names begin with a "$".

ETYP (Entry TYPe) This 10-character string is the type of the entry. Special system types begin with a "$".

When the TOC is initialized, the module that requested the creation of the Workspace defines the size of the character and numeric areas in the Workspace. Four active rows are initially defined in the new TOC. The first two rows are used by the TOC for its own self description, since the TOC storage itself requires a numeric and a character entry. The next two rows describe the initial free space. The initialized TOC configuration is shown below.

EID	ELEN	ELOC	ESOB	ENOB	NUL1	NUL2	ENAM	ETYP
1	7n	1	1	7n	--	--	TOC	NUMERIC
-2	20n	1	1	20n	--	--	TOC	CHARACTER
3	-(lnum-7n)	7n+1	1	lnum-7n	--	--	FREESPACE	NUMERIC
-4	-(lchr-20n)	20n+1	1	lchr-20n	--	--	FREESPACE	CHARACTER
0	--	--	--	--	--	--		
	etc.							

The number of rows in the TOC is 'n', the size in numeric units of the numeric area is 'lnum', and the size in character units of the character area is 'lchr'. Rows 5 through 'n' are initially marked as unassigned, with EID = 0. The TOC can be increased or decreased in size as required. The Workspace Manager keeps track of frequently needed items of information about the TOC such as 'n' or the currently available number of unassigned rows.

Users should always go through an IAC utility to modify or access the TOC, or the TOC may become corrupted.

Workspace Memory Allocation - IACWAM, IACWAB

SUBROUTINE IACWAM (ENAM, ETYP, UFLAG, ELEN, ELOC, EID)

IAC Workspace Allocate Main entry
Purpose: Allocate an unallocated (freespace) area of a given size. An unassigned row in the TOC is filled in with argument values and values supplied from the free space as determined by the Workspace manager. The allocation fails if a FREESPACE entry in the TOC cannot accommodate the allocation request.
TOC columns ESOB and ENOB are set to 1 and ELEN, respectively.

ENAM	(in)	Name of new entry.
ETYP	(in)	Type of new entry.
UFLAG	(in)	Storage units flag (N for numeric, C for character).
ELEN	(in)	Area length needed in UFLAG storage units.
ELOC	(out)	Starting location in Workspace area (0 if failed).
EID	(out)	Entry ID assigned to the new entry (0 if failed).

SUBROUTINE IACWAB (ENAM, ETYP, UFLAG, FRAC, ELEN, ELOC, EID)

IAC Workspace Allocate Big entry
Purpose: Allocate an area as large as the greater of a given size, and a given fraction of the largest unallocated (freespace) area. An unassigned row in the TOC is set to values from the arguments and values determined by the workspace manager from the FREESPACE entries in the TOC. The ESOB and ENOB values are 1 and ELEN, respectively. The allocation fails if a FREESPACE entry in the TOC cannot accommodate the allocation request.

ENAM	(in)	The new entry's name, put in ENAM in the TOC.
ETYP	(in)	The new entry's type, put in ETYP in the TOC.
UFLAG	(in)	Indicator for the entry's storage units, N if numeric or C if character.
FRAC	(in)	The requested fraction of the largest area in FREESPACE.
ELEN	(in)	The minimum required area length, in storage units.
	(out)	The actual length allocated (0 if failed).
ELOC	(out)	The starting location in the Workspace area (0 if failed).
EID	(out)	The new entry's entry ID (0 if failed).

Workspace Memory Deallocation - IACWED

SUBROUTINE IACWED (EID)

IAC Workspace Entry Deallocate
Purpose: Deallocate a given entry, returning its storage to free space. This must not be used on subentries or the entries containing them.

EID	(in)	An allocated entry's ID, to be deallocated.
	(out)	0 if deallocation accomplished, otherwise unchanged.

Workspace TOC Manipulation - IACWAS, IACWEE, IACWER

SUBROUTINE IACWAS (EID, ELEN, ESOB, ELOC)

IAC Workspace Attribute Set
Purpose: The length and/or block size of an entry in the Workspace may be altered by the user. This must not be used on subentries or the entry containing them.

EID	(in)	The existing entry's entry ID.
ELEN	(in)	The new entry length in the Workspace.
ESOB	(in)	The new size of block for the entry. If zero, the Workspace manager may alter the block size as it sees fit, retaining the sign of the prior ESOB value.
ELOC	(out)	The entry's location in the workspace, changed if a COPY operation was required, or 0 if the set operation failed.

SUBROUTINE IACWEE (EID1, EID2)

IAC Workspace Entry Exchange
Purpose: Exchange ID's between two given entries. All TOC attributes are swapped between the two EID's, which means all row values except in the EID column are exchanged.

EID1	(in)	One entry ID.
EID2	(in)	Another entry ID.

SUBROUTINE IACWER (EID, ENAM, ETYP)

IAC Workspace Entry Rename
Purpose: Change the name and type of a Workspace entry.

EID	(in)	Existing entry's ID.
ENAM	(in)	The new string to go into the ENAM column in the TOC.
ETYP	(in)	The new string to go into the ETYP column in the TOC.

Workspace Entry Description Retrieval - IACWAR, IACWEA, IACWEI, IACWEL

SUBROUTINE IACWAR (EID, ELEN, ESOB, ELOC)

IAC Workspace Attribute Retrieval
Purpose: Based on the EID, retrieve an entry's length, block size, and location attributes.

EID	(in)	Existing entry's ID.
ELEN	(out)	Length of entry in storage units (0 if not found).
ESOB	(out)	Size of entry's blocks, less than 0 if the Workspace manager may not decrease the number of blocks that the entry has (0 if not found).
ELOC	(out)	Starting location of the entry (0 if not found).

SUBROUTINE IACWEA (EID, ELEN, ELOC, ENAM, ETYP)

IAC Workspace Entry Attributes
Purpose: Based on the EID, retrieve an entry's length, location, name and type attributes.

EID	(in)	Existing entry's ID.
ELEN	(out)	Length of entry in storage units (0 if not found).
ELOC	(out)	Starting location of the entry (0 if not found).
ENAM	(out)	Name of the entry (blank if not found).
ETYP	(out)	Type of the entry (blank if not found).

INTEGER FUNCTION IACWEI (ENAM, ETYP, EIDEID)

IAC Workspace Entry Identifier
Purpose: Return the entry ID based on the name and type, searching a given list of EID's in an entry whose EID is known.

ENAM	(in)	Expected name of entry.
ETYP	(in)	Expected type of entry.
EIDEID	(in)	EID of an entry which consists of a list of EID's to be searched.
IACWEI	(out)	The entry's EID (0 if no unique one found).

SUBROUTINE IACWEL (ENAM, ETYP, EIDEID, ELEN, ELOC, EID)

IAC Workspace Entry Locate
Purpose: Based on an entry's name, type and a list of EID's to search among, retrieve the entry length, location and ID.

ENAM	(in)	Name of entry.
ETYP	(in)	Type of entry.
EIDEID	(in)	Entry ID of a list of EID, one of which should be an entry with the given name and type.
ELEN	(out)	Length in storage units of entry (0 if failed).
ELOC	(out)	Entry location in the Workspace (0 if failed).
EID	(out)	Entry ID for the found entry (0 if failed).

Workspace Subentry Assignment and Deassignment - IACWDE, IACWSA

SUBROUTINE IACWDE (EID)

IAC Workspace Deassign Entry
Purpose: Deassign an entry which may intersect or overlap with entries that are still active, deallocating any areas no longer referenced by any active entry.

EID	(in)	Existing entry or subentry ID.
	(out)	Zero if the deassign succeeds, otherwise left alone.

SUBROUTINE IACWSA (ENAM, ETYPE, EEID, ELEN, ELOC, EID)

IAC Workspace Subentry Assignment
Purpose: Assign a specified subarea within an existing allocated area as an entry which can be handled with the normal entry manipulating utilities. DO NOT DEALLOCATE these subentries with subroutine IACWED; instead DEASSIGN them with IACWDE.

ENAM	(in)	The new subentry's name.
ETYP	(in)	The new subentry's type.
EEID	(in)	The existing entry's ID over which the subentry will be laid.
ELEN	(in)	The new subentry's length.
ELOC	(in)	The new subentry's starting location, which must be within the area of the existing entry which the subentry intersects.
EID	(out)	The entry ID granted to the subentry.

6.2 DATABASE/FILE MANAGEMENT UTILITIES

A Brief Review of the Database - The Database is stored in, and is identified by the spec of, a host directory or subdirectory. A Database consists of a number of individual host files, two of which (the activities file and the catalog file) have special significance.

The activities file (host spec IACACT.IAC) is a sequential formatted file, used to control multi-user concurrent access to the Database; the first four records provide some general information for the overall Database, and each following record defines one of the current activities. The catalog file (host spec IACCAT.IAC) is a sequential binary file, used to describe and access the IAC (cataloged) files in the Database; it has the form of a relation (table), and contains one row for each IAC file.

Each IAC file may consist of two host files - a data file and an associated textual information file. The data file may be 'structured,' having the IAC class of RELATION, ARRAY or USER; alternatively it may be 'unstructured,' having the IAC class of ANY.

Database Catalog Description - The Database catalog contains the following attributes (columns).

Name	Type	Format	Description
NAME	C10	1X,A10	IAC file name
NUMBER	I1	1X,I10	IAC file number
TYPE	C10	1X,A10	IAC file type
VERSION	I1	1X,I10	IAC file version
CLASS	C10	1X,A10	RELATION, ARRAY, USER, or ANY
STATUS	C1	1X,A1	N=normal, A=archived, I=incomplete
DATASPEC	C20	1X,A20	Data file host spec
TEXTSPEC	C20	1X,A20	Text file host spec
DATE	C8	1X,A8	Date cataloged
TIME	C11	1X,A11	Time cataloged
OWNER	C10	1X,A10	Owner (creator)
MODULE	C10	1X,A10	Creating module
PO	C5	1X,A5	Owner privileges (RWDAL)
PN	C5	1X,A5	Non-owner privileges (RWDAL)
KEYWORDS	C50	/1X,A50	Keywords identification string
TITLE	C80	/1X,A80	Title identification string

Values in the type column above refer to the IAC defined data type (I denotes integer, C denotes character, trailing integer denotes number of numeric or character storage units).

The NAME, NUMBER, TYPE and VERSION attributes comprise the IAC file spec, which has the general form name:number.type;version.

The DATE attribute has the form yy:mm:dd where yy is the year value 00 to 99, mm is the month value 01 to 12, and dd is the day value 01 to 31. TIME has the form hh:mm:ss.cc where hh is the hour value 00 to 23, mm is the minute value 00 to 59, ss is the second value 00 to 59, and cc is the hundredth second value 00 to 99.

PO and PN define the access privileges available to the owner and nonowner, respectively, of the IAC file. They are packed character values: R indicates privilege to read; W indicates write; D indicates delete; A indicates archive and unarchive; and L indicates lock, i.e. the privilege to lock the file against all types of access.

KEYWORDS and TITLE are used for file identification and retrieval purposes. They may contain any sequence of characters.

<u>Database Open/Close Utilities</u> - IACOPE, IACCLO

SUBROUTINE IACOPE (DBSPEC, USER, STATUS, ACCESS) (not implemented)

IAC OPEn database
Purpose: Open an existing or new database.

DBSPEC	(in)	Database spec. This is a character string which defines the spec of an existing host directory or subdirectory where the database resides. Only one database per directory or subdirectory is permitted. Opening a database causes any currently open database to be closed automatically.
USER	(in)	A maximum 10-character string which defines the user name. This name becomes the owner when a new database is opened.
STATUS	(in)	A keyvalue string equal to OLD or NEW (or an abbreviation thereof). OLD opens an existing database, and NEW initializes and opens a new database.

ACCESS (in) A keyvalue string equal to CONCURRENT or NOCONCURRENT (or an abbreviation thereof). NOCONCURRENT provides more efficient access, but does not support concurrent users or job tasks.

SUBROUTINE IACCLO (not yet implemented)

IAC CLOse database
Purpose: Close a currently open database.

File Lock/Unlock Utilities - IACLOC, IACUNL

SUBROUTINE IACLOC (FSPECS, ACT) (not yet implemented)

IAC LOCk activity
Purpose: Enter a username-and-jobid-associated activity into the activities file of the currently open database. The activity may relate to the overall database, to the database catalog, or to one of the cataloged IAC files.

FSPECS (in) A character string containing either a list of IAC file specifications (the specs are separated by commas) ; $CATALOG to indicate an activity on the catalog; or $DATABASE to indicate an activity on the overall database.

ACT (in) A 1-character string equal to either R (read activity), W (write), D (delete), A (archive), L (lock against all activities), C (concurrent open), or N (no-concurrent open).

SUBROUTINE IACUNL (FSPECS, ACT) (not yet implemented)

IAC UNLock activity
Purpose: Remove a username-and-jobid-associated activity from the activities file of the currently open database. The activity may relate to the overall database, to the database catalog, or to one of the cataloged IAC files.

FSPECS (in) A character string containing either a list of IAC file specifications (the specs are separated by commas and may include wild card asterisks to accept all existing substrings as replacement for the *); $CATALOG (or an abbreviation) to indicate an activity on the catalog; or $DATABASE (or an abbreviation) to indicate an access (open) activity on the overall database.

ACT (in) A 1-character string equal to either C (concurrent access), N (noconcurrent access), R (read activity), W (write), D (delete), A (archive), L (lock), or * (all activities).

File Archive/Unarchive Utilities -- IACARC, IACUNA

SUBROUTINE IACARC (QUAL, FSPECS)

IAC ARChive files listed in catalog

Purpose: Archive IAC files, each consisting of a host data file and/or an associated host text file, in the currently open database. The effect of executing this utility is to change the catalog STATUS attribute for each specified file to A (archived). The other catalog attributes remain unchanged, and the file remains in the database directory. The user has responsibility for physically removing the file from the database directory and disposing of the file, after archiving.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s), and N (default) does not.
FSPECS	(in)	The IAC file specifications of the files to be archived. Wild card asterisks may be used within any part of the spec to indicate 'all existing'.

SUBROUTINE IACUNA (QUAL, FSPECS)

IAC UNArchive files listed in catalog

Purpose: Unarchive IAC files, each consisting of a host data file and/or an associated host text file, in the currently open database. The effect of executing this utility is to change the catalog STATUS attribute for each specified file to N (normal). The other catalog attributes remain unchanged, and the file remains in the database directory. The user has responsibility for physically providing the file in the database directory, before unarchiving.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s).
FSPECS	(in)	The IAC file specifications of the files to be unarchived. Wild card asterisks may be used within any part of the spec to indicate 'all existing'.

File Catalog/Uncatalog Utilities - IACCAT, IACUNC

SUBROUTINE IACCAT (QUAL, FSPEC, FCLASS, EIDATT)

IAC CATalog file into database
Purpose: Catalog an IAC file, consisting of a data host file and/or an associated text host file, into the currently open database. The data and/or text host files must exist in the database directory before cataloging.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s), and N (default) does not.
FSPEC	(in)	The IAC file specification of the file to be cataloged. No asterisk wild cards may be used.
FCLASS	(in)	The IAC file class, equal to either ARRAY, RELATION, USER or ANY.
EIDATT	(in)	The entry ID of a 180-character string of user-defined catalog attributes DATASPEC, TEXTSPEC, PO, PN, KEYWORDS and TITLE. See Section 5. A blank value for DATASPEC or TEXTSPEC indicates that the respective host data or text file is absent. An "*" value for PO or PN results in the generation of a database dependent default value.

SUBROUTINE IACUNC (QUAL, FSPECS)

IAC UNCatalog file from database
Purpose: Uncatalog an IAC file, consisting of a data host file and/or an associated text host file, from the currently open database. The data and/or text host files remain in the database directory after uncataloging.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s), and N (default) does not.
FSPECS	(in)	The IAC file specifications of the files to be uncataloged. Wild card asterisks may be used within any part of a spec to indicate 'all existing'.

File Delete/Purge/Rename Utilities - IACDEL, IACPUR, IACREN

SUBROUTINE IACDEL (QUAL, FSPECS)

IAC DELete files from database
Purpose: Delete IAC files, each consisting of a host data file and/or an associated host text file, from the currently open database. The file attributes are deleted from the database catalog, and the files are deleted from the database directory.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s), and N (default) does not.
FSPECS	(in)	The IAC file specifications of the files to be deleted. Wild card asterisks may be used within any part of a spec to indicate 'all existing'.

SUBROUTINE IACPUR (QUAL, FSPECS) (not yet implemented)

IAC PURge files from database
Purpose: Delete all but the highest version of each specified IAC file, from the currently open database. The file attributes are deleted from the database catalog, and the files are deleted from the database directory.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s), and N (default) does not.
FSPECS	(in)	The IAC file specifications of the files to be purged. Wild card asterisks may be used within any part of a spec to indicate 'all existing'.

SUBROUTINE IACREN (QUAL, FSPECS, RFSPEC)
IAC RENname file in database
Purpose: Change IAC file specifications, in the currently open database.

QUAL	(in)	A 1-character string equal to either V (verify) or N (noverify). V causes a verification message to be written to the IAC message file(s), and N (default) does not.
FSPECS	(in)	The existing IAC file specifications of the files to be renamed. Wild card asterisks may be used within any part of the spec to indicate 'all existing'.
RFSPEC	(in)	The new IAC file specification for the files to be renamed. A wild card asterisk may be used for any part of the spec to indicate 'same as corresponding part in existing spec'.

6.3 FILE TRANSFER UTILITIES

A Brief Review of the IAC Data Areas and Files - There are three data areas available to IAC modules: the IAC Database, the IAC Workspace, and the Host File system. The file transfer utilities are provided to move data among these different areas.

An IAC file logically consists of a data file and/or a narrative text file. IAC data files are designated as either 'structured' or 'unstructured', depending upon their contents. A structured file belongs to one of the three classes RELATION, ARRAY or USER; it contains logical data groupings called 'entries', some of which may be defined according to IAC standard formats. An unstructured file belongs to class ANY; it can be anything--an executable module, command procedure, log file, or even a structured file--that an IAC user may need to reference.

The IAC workspace file transfer utilities are IACGET and IACPUT. These utilities handle all classes of IAC files. The first will get a file from the IAC Database or the Host File system, and transfer it to either the IAC Workspace or the Host File system, depending on the type of file ('structured' or not). The other reverses the transfer. Qualifying options are provided for the verification of successful completion, selection of the area from/to which the file is to be transferred, and for selection of data and/or accompanying descriptive text.

IACREA and IACWRI are user storage file transfer utilities. These utilities handle all classes of IAC files. The first will get a file from the IAC Database or the Host File system, and transfer it to user provided storage. The other reverses the transfer. Qualifying options are provided as for IACGET and IACPUT.

Finally, IACGAR and IACPAR are similar to IACREA and IACWRI. These utilities handle RELATION and ARRAY classes of IAC files. The calling interface is simpler than that for IACREA and IACWRI, but only constant type attributes are handled.

IAC Workspace File Transfer Utilities - IACGET, IACPUT

SUBROUTINE IACGET (QUAL, FSPEC, FCLASS, ESPECS, EIDEID, EIDATT)

IAC GET file to IAC workspace.

Purpose: Copy an IAC structured data file, from a host directory or an IAC database, to the IAC workspace. The text file from an IAC database file may also be copied to the default directory.

QUAL	(in)	A character string of 'keyname=value' qualifiers separated by commas. The available qualifiers are described in Table 6.3-1.
FSPEC	(in)	The spec of the host file (may include directory etc.) or IAC database file.
FCLASS	(in)	The expected file class (ARRAY, RELATION or USER). If the value is non-blank, it is checked against the actual class.
	(out)	The actual file class.
ESPECS	(in)	A character string containing entry 'name.type' specifications, separated by commas if there are more than one. These entries are selected from the structured file. Asterisks can be used within the name or type as 'wild cards' to accept all existing substrings as replacement for the *.
EIDEID	(out)	The entry ID of a list of all copied entry ID's for a structured file. The length of this entry is equal to the number of retrieved entries. If no entries are retrieved, the entry pointed to by EIDEID has length 0.
EIDATT	(out)	The entry ID of a 180-character string of user defined catalog attributes DATASPEC, TEXTSPEC, PO, PN, KEYWORDS and TITLE. See Section 5.1.1. If the file is copied from a host directory, the string is blank. If the file is copied from an IAC database, the string contains values from the database catalog; if the data or text files are not copied (see QUAL argument), then DATASPEC or TEXTSPEC is set to blank.

Note: IACGET may also be used to copy an IAC unstructured data file (class ANY). For FCLASS (in) of ANY, ESPECS is ignored. For FCLASS (out) of ANY, EIDEID is set to zero and the data and/or text files are copied to the default directory.

SUBROUTINE IACPUT (QUAL, FSPEC, FCLASS, ESPECS, EIDEID, EIDATT)

IAC PUT file from IAC workspace.

Purpose: Copy an IAC structured data file, from the IAC workspace to a host directory or an IAC database. The text file from IAC database file may also be copied from the default directory.

QUAL	(in)	A character string of keyname=value qualifiers separated by commas. The available qualifiers are described in Table 6.3-1.
FSPEC	(in)	The spec of the host file (may include directory etc.) or IAC database file.
FCLASS	(in)	The file class (ARRAY, RELATION or USER).
ESPECS	(in)	A character string containing entry 'name.type' specifications, separated by commas if there are more than one. These entries are selected from the entries pointed to by the EIDEID argument. Asterisks can be used within the name or type as 'wild cards' to accept all existing substrings as replacement for the *.
EIDEID	(in)	The entry ID of a list of copiable entry ID's (zeros in the list are ignored) for a structured file.
EIDATT	(in)	The entry ID of a 180-character string of user defined catalog attributes DATASPEC, TEXTSPEC, PO, PN, KEYWORDS and TITLE. See Section 5.1.1. If the file is copied to a host directory, EIDATT is not currently used. If the file is copied to an IAC database, the value of "*" for DATASPEC, PO or PN results in the generation of a database dependent default for insertion into the catalog. If the data or text file is not copied (see the QUAL argument), a blank value is inserted into the catalog.

Note: IACPUT may also be used to copy an IAC un-structured data file (class ANY). For FCLASS (in) of ANY, the arguments ESPECS and EIDEID are ignored; and the data and/or text files are copied to the default directory.

V=V N null	This qualifier specifies the verification option. Value V requests that a message be written indicating whether the file transfer was successful; N does not; and null (default) is equivalent to N.
A=H D HD DH null	This qualifier specifies the area from/to which the file is to be transferred. H specifies host directory; D specifies database; HD specifies host directory if possible, else database; DH specifies database if possible, else host directory; and null (default) is equivalent to HD. For example, a file read with an A=DH specification results in an attempt to retrieve the file from the IAC database; if retrieval is not successful (e.g. database not open, invalid IAC file spec, file not found, or error during read), an attempt is then made to retrieve the file from the host directory.
D=D * N null	If the file is being transferred to or from the host directory, this qualifier is ignored. If the file is being transferred to or from an IAC database, the value D specifies that the data file within the IAC file is to be transferred, keeping the existing host file spec; * indicates that the text file will be transferred but that a new host file spec will be generated if necessary; N indicates that the text file is not to be transferred; and null (default) is equivalent to *.
T=T * N null	If the file is being transferred to or from the host directory, this qualifier is ignored. If the file is being transferred to or from an IAC database, the value T specifies that the text file within the IAC file is to be transferred, keeping the existing host file spec; * indicates that the text file will be transferred but that a new host file spec will be generated if necessary; N indicates that the text file is not to be transferred; and null (default) is equivalent to N.

Table 6.3-1. Qualifiers for File Transfer Utilities

User-Storage File Transfer Utilities - IACREA, IACWRI, IACGAR, IACPAR

SUBROUTINE IACREA (QUAL, FSPEC, FCLASS, ESPECS, LTABLE, ETABLE, UCATT
, LNAR, NAR, LCAR, CAR)

IAC REAd file to user storage

Purpose: Copy an IAC structured data file, from a host directory or an IAC database to user provided storage. The text file from an IAC database file may also be copied to the default directory.

QUAL	(in)	A character string of 'keyname=value' qualifiers separated by commas. The available qualifiers are described in Table 6.3-1.
FSPEC	(in)	The spec of the host file (may include directory etc.) or IAC database file.
FCLASS	(in)	The expected file class (ARRAY, RELATION, or USER). If the value is non-blank, it is checked against the actual class.
	(out)	The actual file class.
ESPECS	(in)	A character string containing entry 'name.type' specifications, separated by commas if there are more than one. These entries are selected from the structured file. Asterisks can be used within the name or type as 'wild cards' to accept all existing substrings as replacement for the *. If ESPECS is blank, the list of selected entries is defined by the pre-allocated entries in ETABLE.
LTABLE	(in)	A two element integer vector. LTABLE(1) is the maximum size (number of lines) of ETABLE. LTABLE(2) is the current actual size (number of lines) of ETABLE, i.e. the number of entries for which storage has been pre-allocated by the user.
	(out)	LTABLE(1) is unchanged. LTABLE(2) is the final actual size of ETABLE, i.e. the total number of entries retrieved; this is the sum of the number of pre-allocated entries and the number of entries post-allocated by IACREA.
ETABLE	(in)	A table of 80-character lines (an array of 80-character elements). See Table 6.3-2 for ETABLE format. Each line 1 through LTABLE(2) corresponds to a pre-allocated entry, which is to be retrieved based upon the ESPECS specifications: colums 1-40 in each of these lines must be defined and 41-80 may be blank. Other lines are ignored.
	(out)	Lines 1 through input LTABLE(2) (corresponding to pre-allocated entries) are unchanged, except that if the maximum and actual sizes (column 41-60 and 61-80) were blank, the maximum sizes is set to the length (columns 22-30) and the actual sizes is set to the size of the retrieved entry. Each line input LTABLE(2)+1 through output LTABLE(2) corresponds to an additional post-allocated entry, which is retrieved based upon the ESPECS

		specifications; columns 1-80 in each of these lines are defined by IACREA.
UCATT	(out)	A 180-character string of user defined catalog attributes DATASPEC, TEXTSPEC, PO, PN, KEYWORDS and TITLE. See Section 5.1.1. If the file is copied from a host directory, UCATT is set to blank. If the file is copied from an IAC database, UCATT contains values from the database catalog; if the data or text files are not copied (see QUAL argument) then DATASPEC or TEXTSPEC is set to blank.
LNAR	(in)	The maximum length (last available storage location) of the array NAR. Note that length is measured in numeric storage units.
	(out)	The actual length (last used storage location) of the array NAR.
NAR	(in)	User provided numeric storage array.
LCAR	(in)	The maximum length (last available storage location) of the array CAR. Note that length is measured in character storage units.
	(out)	The actual length (last used storage location) of the array CAR.
CAR	(in)	User provided character storage array.

Note: IACREA may also be used to copy an IAC unstructured data file (class ANY). For FCLASS (in) of ANY, the input arguments ESPECS, LTABLE, ETABLE, LNAR, NAR, LCAR and CAR are ignored. For FCLASS (out) of ANY, there is no output for arguments LTABLE, ETABLE, LNAR, and LCAR; and the data and/or text files are copied to the default directory.

SUBROUTINE IACWRI (QUAL, FSPEC, FCLASS, ESPECS, LTABLE, ETABLE, UCATT
, LNAR, NAR, LCAR, CAR)

IAC WRIte file from user storage

Purpose: Copy an IAC structured data file, from user provided storage to a host directory or an IAC database. The text file for an IAC database file may also be copied from the default directory.

QUAL	(in)	A character string of 'keyname=value' qualifiers separated by commas. The available qualifiers are described in Table 6.3-1.
FSPEC	(in)	The spec of the host file (may include directory etc.) or IAC database file.
FCLASS	(in)	The file class (ARRAY, RELATION, or USER).
ESPECS	(in)	A character string containing entry 'name.type' specifications, separated by commas if there are more than one. These entries correspond to the lines within ETABLE. Asterisks can be used within the name or type substrings as 'wild cards' to accept all existing substrings as replacement for the *. If ESPECS is blank, the list of selected entries corresponds to all entries in ETABLE (i.e. blank is equivalent to "*.*")
LTABLE	(in)	A two element integer vector. LTABLE(1) is the maximum size (number of lines) of ETABLE. LTABLE(2) is the number of allocated entries.
	(out)	LTABLE(1) is unchanged. LTABLE(2) is the number of entries that were written, based upon the ESPECS specifications.
ETABLE	(in)	A table of 80-character lines (an array of 80-character elements). See Table 6.3-2 for ETABLE format. Each line 1 through LTABLE(2) defines an entry in the user provided storage arrays (NAR and CAR). If the maximum and actual sizes (columns 41-60 and 61-80) are blank, then the entry size is the length (columns 22-30).
UCATT	(in)	A 180-character string of user defined catalog attributes DATASPEC, TEXTSPEC, PO, PN, KEYWORDS and TITLE. See Section 5.1.1. If the file is copied to a host directory, UCATT is not currently used. If the file is copied to an IAC database, the value of "*" for DATASPEC, PO or PN results in the generation of a database dependent default for insertion into the catalog. If the data or text file is not copied (see the QUAL argument), a blank value is inserted into the catalog.
LNAR	(in)	The maximum length (last available storage location) of the array NAR. Note that length is measured in numeric storage units.
NAR	(in)	User provided numeric storage array.
LCAR	(in)	The maximum length (last available storage location) of the array CAR. Note that length is measured in character storage units.
CAR	(in)	User provided character storage array.

Note: IACWRI may also be used to copy an IAC unstructured data file (class ANY). For FCLASS of ANY, the arguments ESPECS, LTABLE, ETABLE, LNAR, NAR, LCAR and CAR are ignored; and the data and/or text files are copied from the default directory.

Entity	Columns	Format	Description
name	1:10	A10	entry name
type	11:20	A10	entry type
unit	21:21	A1	storage flag (C or N)
length	22:30	I9	maximum total size
location	31:40	I10	starting location
maximum sizes	41:60	A20	maximum index sizes
actual sizes	61:80	A20	actual index sizes

Notes:
1) Maximum and actual sizes may be used to define the unpacked storage characteristics of an entry; these sizes are represented as a left-adjusted string of integers, separated by commas. Number of actual sizes must equal number of maximum sizes, and each actual size must be less than or equal to the corresponding maximum size. The length is the product of the maximum sizes. All sizes are measured in terms of numeric or character storage units.
2) IACREA allows the last actual index size to be "*" on input. On output, IACREA will replace the "*" with the actual size.
3) IACREA allows the length to be negative on input. The negative length indicates an optional entry. If this entry is found in the file, IACREA returns the length as a positive value.

Table 6.3-2: ETABLE Format for IACREA and IACWRI Utilities

SUBROUTINE IACGAR (QUAL, FSPEC, LNAR, NAR, LCAR, CAR, ATTS, LOCS
, NUMDIM, MAXDIM, ACTDIM, UCATT)

IAC Get Array or Relation to user storage

Purpose: Copy an IAC ARRAY or RELATION data file, from a host directory or an IAC database to user provided storage. The text file from an IAC database file may also be copied to the default directory.

QUAL	(in)	A character string of 'keyname=value' qualifiers separated by commas. The available qualifiers are described in Table 6.3-1.
FSPEC	(in)	The spec of the host file (may include directory etc.) or IAC database file.
LNAR	(in)	Length (numeric storage units) of NAR.
NAR	(in)	Array for storing retrieved numeric attributes.
LCAR	(in)	Length (character storage units) of CAR.
CAR	(in)	Array for storing retrieved character attributes.
ATTS	(in)	A character string of 'name,type,index' item triplets specifying attributes to be retrieved. The items and triplets may be separated by commas and/or slashes. An integer number for the name requests retrieval of the ith attribute associated with the specified index.
LOCS	(in)	A vector of starting locations within NAR and/or CAR, one for each attribute. Locations are defined in terms of numeric and/or character storage units.
NUMDIM	(in)	For a RELATION, NUMDIM is ignored. For an ARRAY, ABS(NUMDIM) is the number of dimensions: 1) NUMDIM>0 indicates that any arrays (index 0 attributes) are to be stored in unpacked form. 2) NUMDIM<0 indicates that any arrays are to be stored in packed form.
MAXDIM	(in)	MAXDIM is ignored except when returning an array (index 0) attribute of an ARRAY with NUMDIM>0. In this case it is a vector of maximum dimensioned index sizes for the internal array, and defines the unpacked storage characteristics.
ACTDIM	(out)	For a RELATION, ACTDIM(1) is the relation size (number of tuples). For an ARRAY, ACTDIM is a vector of actual sizes for ARRAY index 1, 2, etc.
UCATT	(out)	As for IACGET and IACREA.

SUBROUTINE IACPAR (QUAL, FSPEC, LNAR, NAR, LCAR, CAR, ATTS, LOCS
, NUMDIM, MAXDIM, ACTDIM, UCATT)

IAC Put Array or Relation to user storage

Purpose: Copy an IAC ARRAY or RELATION data file, from user provided storage to a host directory or an IAC database. The text file for an IAC database file may also be copied from the default directory.

QUAL	(in)	A character string of 'keyname-value' qualifiers separated by commas. The available qualifies are described in Table 6.3-1.
FSPEC	(in)	The spec of the host file (may include directory etc.) or IAC database file.
LNAR	(in)	Length (numeric storage units) of NAR.
NAR	(in)	Array contained numeric attributes to be sent.
LCAR	(in)	Length (character storage units) of CAR.
CAR	(in)	Array containing character attributes to be sent.
ATTS	(in)	A character string of 'name,type,index' item triplets specifying attributes to be retrieved. The items and triplets may be separated by commas and/or slashes.
LOCS	(in)	A vector of starting locations within NAR and/or CAR, one for each attribute. Locations are defined in terms of numeric and/or character storage units.
NUMDIM	(in)	For a RELATION, NUMDIM must be 0. For an ARRAY, ABS(NUMDIM) is the number of dimensions: 1) NUMDIM>0 indicates that any arrays (index 0 attributes) are stored in unpacked form. 2) NUMDIM<0 indicates that any arrays are stored in packed form.
MAXDIM	(in)	MAXDIM is ignored except when sending an array (index 0) attribute of an ARRAY with NUMDIM>0. In this case it is a vector of maximum dimensioned index sizes for the internal array, and defines the unpacked storage characteristics.
ACTDIM	(out)	For a RELATION, ACTDIM(1) is the relation size (number of tuples). For an ARRAY, ACTDIM is a vector of actual sizes for ARRAY index 1, 2, etc.
UCATT	(out)	As for IACPUT and IACWRI.

Note: Currently IACPAR requires some extra space to be provided at the end of CAR, for its own use. The required space is (50) times (number of attributes plus number of indexes having associated attributes) characters; e.g. 250 characters for an array containing 1 index-0 attribute and 2 index-1 attributes. This requirement is expected to be deleted from IAC Level 2.

6.4 UTILITY CONDITION PROCESSING

A Brief Review of Condition Information Available - The IAC utilities recognize four levels of message/error conditions. Conditions of level 0 (message prefixed with *INFORMATION*) are informational and do not result from any type of error. The other three condition levels correspond to an error level, generally having the following respective characteristics. Level 1 (message prefixed with *WARNING*) is an error that permits a limited continuation of the function and does no damage to recorded data. Level 2 (message prefixed with *ERROR*) is more severe; the function cannot continue, and is prevented from damaging recorded data. Level 3 (message prefixed with *ABORT*) occurs when an unrecoverable error happens and the module must be aborted.

Three common blocks (IACMEC, IACMAD and IACOUT) contain a number of IAC integer variables affecting, or resulting from, the condition handling process. The variables within these common blocks are available via the utilities IACOND and IACINT. (See Section 6.5). For example, the statement

```
IERR = IACOND ( )
```

sets the value of the user variable IERR to the current IAC condition level. (It is usually advisable to check the condition level after a call to other IAC utilities). The statement

```
CALL IACINT ('SET', 'CLEVEL', 0)
```

would set the condition level to zero, and the statement

```
CALL IACINT ('GET', 'NWARN', N)
```

would get the value of the current number of IAC warning conditions and assign it to the user variable N.

In order to make IAC coding interfaces visible, and to maintain upward compatibility of application software with future IAC releases, it is

recommended that the IAC variables be accessed only via these utilities. However, for purposes of information and possible debugging, the common blocks are defined below. (See utilities IACOND and IACINT for definitions of the variables).

```
COMMON /IACMEC/CLEVEL
COMMON /IACMAD/NWARN, NERROR, NABORT, WARMAX, ERRMAX, PRINTI, PRINTW
COMMON /IACOUT/MESOUT, ALTOUT
```

Condition Level Effects - In general, if a utility is entered with CLEVEL nonzero, an immediate return occurs. However, the following utilities attempt to perform their functions regardless of CLEVEL.

IACEND	IACHOS	IACINT	IACMES	IACOND
IACWAR	IACWEA	IACWED	IACWEE	IACWER

Message Utility - IACMES

```
SUBROUTINE IACMES (LEVEL, MESSAG)
```

IAC MESsage recording

Purpose: Set the IAC condition level, i.e. the CLEVEL variable. Display the message, subject to control by the IAC variables PRINTI, PRINTW, MESOUT and/or ALTOUT. See utilities IACOND and IACINT.

LEVEL (in) The message level or its negative:
- 0 for information - no error has occurred
- 1 for warning - a probably limiting but nondestructive error has occurred
- 2 for error - a serious error has occurred
- 3 for abort - the run cannot continue

If LEVEL is positive and IACMES detects an abort situation (determined by values for LEVEL, WARMAX and/or ERRMAX), IACEND is called to force a module exit. If LEVEL is negative and IACMES detects an abort situation, a return is made to the calling routine.

MESSAG (in) The message to be displayed. It will be preceded by *INFORMATION*, *WARNING*, *ERROR*, or **ABORT** for levels 0, 1, 2 or 3, respectively. Up to 80 characters will be put on a line unless the string contains binary zeros (created with CHAR(0)), in which case the lines are displayed, separated at the zeros and aligned with the message beyond the *prefix*.

6.5 MISCELLANEOUS IAC UTILITIES

This section describes miscellaneous IAC utilities which enable the use of other utilities or which provide general purpose support functions.

Initiation/Termination IAC Utilities - IACBEG, IACEND

SUBROUTINE IACBEG (MODNAM, QUAL, FUNITS, WUNITS, NSIZE, CSIZE)

IAC BEGin

Purpose: Initiate communication with IAC utilities. The initiation process depends upon the arguments to IACBEG and the values on the parameter file.

MODNAM (in) A string containing the module name (up to ten characters using underscores and upper-case letters).

QUAL (in) A string containing qualifiers separated by commas (no embedded blanks). Each qualifier is of the form "keyname=value". The following is a list of the available keynames and values.

- G=G / N / null — Value G requests reading of GEN type parameters from the parameter file and storing into an IAC provided array (absence of parameter file generates an error); N does not; and null (default) reads the file if it exists. See also the utility IACGEN.
- D=D / N / null — Value D requests opening of the IAC database specified on parameter file (absence of database spec generates an error); N does not; and null (default) opens database if spec exists. See also utility IACOPE.
- W=n / null — Value of integer n gives number of IAC warning conditions prior to automatic abort; and null (default) is no abort on warnings.
- E=n / null — Value of integer n gives number of IAC error conditions prior to automatic abort; and null (default) is no abort on errors.

FUNITS (in) A string containing functional unit definitions separated by commas (no blanks). Each definition is of the form "keyname=value". The value is a FORTRAN unit number (default value null indicates that the function is not active). The module implementor is responsible for assigning the unit to a device or file (e.g. via JCL or a FORTRAN OPEN statement). The assignment may need to be in

effect prior to calling certain utilities. The following is a list of available functions.

O=n — May copy the output (i.e. log file) of
null — any IACHOS spawn process to FORTRAN unit n. For details see utility IACHOS.

M=n — Write IAC messages to FORTRAN message
null — output unit n. See also utility IACMES.

A=n — Write IAC messages to FORTRAN alternate
null — output unit n. If M and A define the same unit, only one message is written. See also utility IACMES.

WUNITS (in) A string containing working unit definitions separated by commas (no blanks). Each definition is a FORTRAN unit number. The working units are used by the IAC utilities to read and write host files. The default string is "30,31,32,33". At least 4 units should be defined.

NSIZE (in) The size in numeric storage units of the IAC numeric workspace (i.e. common block /IACWAN/). When computing this the user should allow for the IAC workspace TOC and other IAC utility overhead.

CSIZE (in) The size in character storage units of the character workspace (i.e. common block /IACWAC/). The IAC overhead allowance for NSIZE also applies to CSIZE.

SUBROUTINE IACEND (QUAL)

IAC END

Purpose: Terminate communication with IAC utilities. Close the database (if open); display a diagnostic summary (if requested); initialize key IAC variables set by IACBEG to null values; return a host status symbol (if requested); and exit the module (if requested). Note that if an exit from the module does not occur, the value of CLEVEL is kept, and is available via call to IACOND.

QUAL (in) A string containing qualifiers separated by commas (no embedded blanks). Each qualifier is of the form "keyname=value". The following is a list of the available keynames and values.

S=S
N
null — Value S requests a diagnostic message summary; N does not; and null (default) is equivalent to S.

E=E
N
null — Value E requests an exit from the module; N requests a return to the calling routine; and null (default) is equivalent to E.

C=C
c
W
E
A — Value C requests that the IAC variable CLEVEL be translated and returned as a host system status symbol; c is an integer valued string (e.g. "2"), whose value is any value which CLEVEL can

N null	assume, and is returned as for CLEVEL; W, E, and A request that the host status symbol be returned, based upon the maximum severity level defined by the respective IAC variables NWARN, NERROR and NABORT (e.g. E returns an error status if NERROR or NABORT is not zero); N does not return the status symbol; and null (default) is equivalent to N.

Parameter File Handling - IACPFR, IACPFW

SUBROUTINE IACPFR (LPARAM, PARAM, TDLIST)

IAC Parameter File Read

Purpose: Selected lines from a parameter file (usually written via the IAC Executive) are read into a user-provided character array. When IACPFR is called, the parameter file spec is assumed to exist as the value of the host local symbol PARM_FILE_SPEC. The file and the array each consist of a number of card images in the format described in Table 6.5-1.

LPARAM	(in)	The maximum number of card images to be stored.
	(out)	The actual number of card images stored in PARAM.
PARAM	(out)	The card images are written into this local array.
TDLIST	(in)	A string containing 'type.destination' identifiers, separated by commas if there are more than one. All type and destination identifiers in the parameter file will be checked to see if the identifier matches any in TDLIST. Asterisks may be used within the type or destination as 'wild cards' to accept all existing substrings as replacement for the *. The IAC standard identifier types and destinations are defined in Table 6.5-2. Other types and destinations may be used, and might be meaningful to a particular module executed in a standalone mode.

SUBROUTINE IACPFW (LPARAM, PARAM, TDLIST)

IAC Parameter File Write

Purpose: Selected lines from a user-provided character array are written into a parameter file. When IACPFW is called, the parameter file spec is generated via the FORTRAN-like catenation 'I'//runid//'.PPP', where runid is the value of the parameter named RUNID which is assumed to exist in the user-provided character array. The array and the file each consist of a number of card images in the format described by Table 6.5-1.

LPARAM	(in)	The number of card images stored in PARAM.
PARAM	(in)	The card images are read from this local array.
TDLIST	(in)	A character string containing 'type.destination' identifiers, separated by commas if there are more than one. All type and destination identifiers in the local array will be checked to see if the identifier matches any in TDLIST. Asterisks may be used within the type or destination as 'wild cards' to accept all existing substrings as replacement for the *. The IAC standard identifier types and destinations are defined in Table 6.5-2, but others may be used.

Columns	Usage	Format	Description
1:10	Name	A10	
11:15	Type	A5	One of GEN, MOD or RUN.
16:20	Destination	A5	One of ANY, EXE, JCL or SYM.
21:28	unused	8X	
29:30	Length	I2	The length of Value on this card.
31:80	Value/unused	A	All or part of Value, followed by unused columns. The Value may be split and continued on successive cards named CONTINUE. Unused columns may be blank or contain comments etc.

Table 6.5-1: Parameter Card Image Format

Type	Source	Dest	Target
GEN	general parameter	ANY	any destination
MOD	MODULES table parameter	EXE	module executable code
RUN	RUN table parameter	JCL	module JCL procedure
		SYM	host symbol definition table

Table 6.5-2: Parameter Standard Types and Destinations

File I/O Handling - IACUAD

SUBROUTINE IACUAD (QUAL, UNIT)

IAC Unit Assign/Deassign
Purpose: Assign or deassign an I/O unit number, from the pool of working units defined by the WUNITS parameter in the call to IACBEG.

QUAL	(in)	A 1-character string equal to either A (assign) or D (deassign)
UNIT	(in)	Unit number for deassign.
	(out)	Unit number for assign (0 if failed); 0 for deassign (unchanged if failed).

GEN Parameter Retrieval Utility - IACGEN

SUBROUTINE IACGEN (GENNAM, GENLEN, GENVAL)

IAC retrieval of a GEN type parameter
Purpose: Retrieve the current value of a GEN type parameter.

GENNAM	(in)	Name of parameter
GENLEN	(out)	Length of parameter value
GENVAL	(out)	Character variable containing the parameter value.

Host Command Execution Utility - IACHOS

SUBROUTINE IACHOS (COMAND, OUTFIL)

IAC execution of a HOSt (i.e. JCL) command
Purpose: The execution of a host (i.e. operating system) command from within a FORTRAN routine. All variables within the routine (except those associated with this utility) are unchanged after the host command execution.

COMAND	(in)	The host command to be executed
OUTFIL	(in)	Specifies the log file for the host command process. If OUTFIL is non-blank, it gives the spec of a new host file which is to be the log file. If OUTFIL is blank, there are two cases: 1) Interactive run - log file is the terminal. 2) Batch run - A temporary scratch log file is written. If the IACBEG qualifier O=n was specified, the scratch file is appended to unit n and then deleted. If O=null, the scratch file is deleted and its contents are lost.

Assignment and Retrieval of IAC Variables - IACOND, IACINT

INTEGER FUNCTION IACOND ( )

IAC cONDition retrieval
Purpose: Retrieve the IAC condition level.

IACOND (out) The value of the IAC condition level.

SUBROUTINE IACINT (QUAL, INTNAM, INTVAL)

IAC INTeger assignment or retrieval
Purpose: Set or retrieve the value of one of the standard IAC integer variables.

QUAL (in) A string which specifies the operation. GET indicates retrieval, SET indicates assignment.

INTNAM (in) A string equal to the name of the IAC integer variable. Operations are currently supported for the following names.

- CLEVEL The current condition level. If CLEVEL is positive on entering a utility, the utility usually returns without major processing. If CLEVEL is zero or negative, the utility performs its function, and if successful returns without altering CLEVEL. Any unsuccessful utility execution results in CLEVEL being set to a level of 1, 2, or 3; a message is written to the message unit(s) MESOUT and/or ALTOUT.
- NWARN The number of level 1 conditions since initiation of IACBEG.
- NERROR The number of level 2 conditions since initiation of IACBEG.
- NABORT The number of level 3 conditions since initiation of IACBEG.
- WARMAX Maximum number of level 1 conditions. If the utility IACMES recognizes that this limit has been reached, it automatically calls IACEND to cause the module to abort and exit. (If zero or negative, this limit is ignored).
- ERRMAX Maximum number of level 2 conditions. Action by IACMES is the same as for WARMAX.
- PRINTI Flag to indicate whether level 0 (information) messages are to be printed (zero indicates no printing, nonzero indicates printing).
- PRINTW Flag to indicate whether level 1 (warning) messages are to be printed (zero indicates no printing, nonzero indicates printing).

	MESOUT	FORTRAN unit number for messages (if zero, no messages are written).
	ALTOUT	Alternate FORTRAN unit number for messages (if zero, no messages are written). If MESOUT and ALTOUT are equal and nonzero, only one message is written.
	NSIZE	Size (storage units) of numeric workspace.
	CSIZE	Size (storage units) of character workspace.
INTVAL	(in)	Value to be assigned to INTNAM for a SET operation.
	(out)	Value returned in INTVAL for a GET operation

6.6 UTILITY SUMMARY REFERENCE

Message Utilities - The following utility is defined.

```
IACMES  sub (LEVEL,MESSAG)                          Message display
```

File Transfer Utilities - The following 6 utilities are defined.

```
IACGAR  sub (QUAL,FSPEC,LNAR,NAR,LCAR,CAR           Copy to user storage
            ,ATTS,LOCS
            ,NUMDIM, MAXDIM,ACTDIM,UCATT)
 IACPAR sub (QUAL,FSPEC,LNAR,NAR,LCAR,CAR           Copy from user storage
            ,ATTS,LOCS
            ,NUMDIM, MAXDIM,ACTDIM,UCATT)
IACGET  sub (QUAL,FSPEC,FCLASS,ESPECS               Copy to IAC workspace
            ,EIDEID,EIDATT)
IACPUT  sub (QUAL,FSPEC,FCLASS,ESPECS               Copy from IAC workspace
            ,EIDEID,EIDATT)
IACREA  sub (QUAL,FSPEC,FCLASS,ESPECS               Copy to user storage
            ,LTABLE,ETABLE,UCATT
            ,LNAR,NAR,LCAR,CAR)
IACWRI  sub (QUAL,FSPEC,FCLASS,ESPECS
            ,LTABLE,ETABLE,UCATT                    Copy from user storage
            ,LNAR,NAR,LCAR,CAR)
```

Miscellaneous Utilities - The following 9 utilities are defined.

```
IACBEG  sub (MODNAM,QUAL,FUNITS,WUNITS              Begin utility communication
            ,NSIZE,CSIZE)
IACEND  sub (QUAL)                                  End utility communication
IACGEN  sub (GENNAM,GENLEN,GENVAL)                  Retrieve GEN parameter
IACHOS  sub (COMAND,OUTFIL)                         Execute host command
IACINT  sub (QUAL,INTNAM,INTVAL)                    IAC integer assign/retrieve
IACOND  fun ()                                      IAC condition retrieval
IACPFR  sub (LPARAM,PARAM,TDLIST)                   Read parameter file
IACPFW  sub (LPARAM,PARAM,TDLIST)                   Write parameter file
IACUAD  sub (QUAL,UNIT)                             I/O unit assign/deassign
```

Workspace Utilities - The following 12 utilities are defined.

IACWAB	sub	(ENAM,ETYP,UFLAG,FRAC,ELEN ,ELOC,EID)	Allocate big entry
IACWAM	sub	(ENAM,ETYP,UFLAG,ELEN,ELOC ,EID)	Allocate main entry
IACWAR	sub	(EID,ELEN,ESOB,ELOC)	Attribute retrieve
IACWAS	sub	(EID,ELEN,ESOB,ELOC)	Attribute set
IACWDE	sub	(EID)	Deassign subentry
IACWEA	sub	(EID,ELEN,ELOC,ENAM,ETYP)	Retrieve entry attributes
IACWED	sub	(EID)	Entry deallocation
IACWEE	sub	(EID1,EID2)	Entry exchange
IACWEI	fun	(ENAM,ETYP,EIDEID)	Retrieve entry ID
IACWEL	sub	(ENAM,ETYP,EIDEID,ELEN ,ELOC,EID)	Entry locate
IACWER	sub	(EID,ENAM,ETYP)	Entry rename and retype
IACWSA	sub	(ENAM,ETYP,EEID,ELEN,ELOC ,EID)	Subentry assignment

Database/File Management Utilities - The following 11 utilities are defined.

IACARC	sub	(QUAL,FSPECS)	Archive database files
IACCAT	sub	(QUAL,FSPEC,FCLASS,EIDATT)	Catalog database file
*IACCLO	sub	no arguments	Close database
IACDEL	sub	(QUAL,FSPECS)	Delete database files
*IACLOC	sub	(FSPECS,ACT)	Add database activity
*IACOPE	sub	(DBSPEC,USER,STATUS,ACCESS)	Open database
*IACPUR	sub	(QUAL,FSPECS)	Purge database files
IACREN	sub	(QUAL,FSPECS,RFSPEC)	Rename database files
IACUNA	sub	(QUAL,FSPECS)	Unarchive database files
IACUNC	sub	(QUAL,FSPECS)	Uncatalog database files
*IACUNL	sub	(FSPECS,ACT)	Delete database activity

*Not yet implemented.

THIS PAGE INTENTIONALLY LEFT BLANK

7.0 MODULE IMPLEMENTATION

This section describes the approach for implementing modules within IAC. Related information is contained in Section 5 (Data Organization and Storage) and Section 6 (Utilities).

In order to facilite easy reference, three key illustrations are first provided. Figure 7.0-1 shows a typical IAC directory organization; Figure 7.0-2 shows the module execution schematic; and Table 7.0-1 lists the IAC system GEN type parameters. Detailed discussions of these illustrations are included within the following topics of this section.

The first topic defines the IAC system directory organization. The next three topics discuss the definition and use of the IAC global symbol, the module execution sequence which the IAC system employs when a module is invoked, and additional general software tools which IAC provides to the module implementor. The last three topics deal with the definition and handling of module parameters.

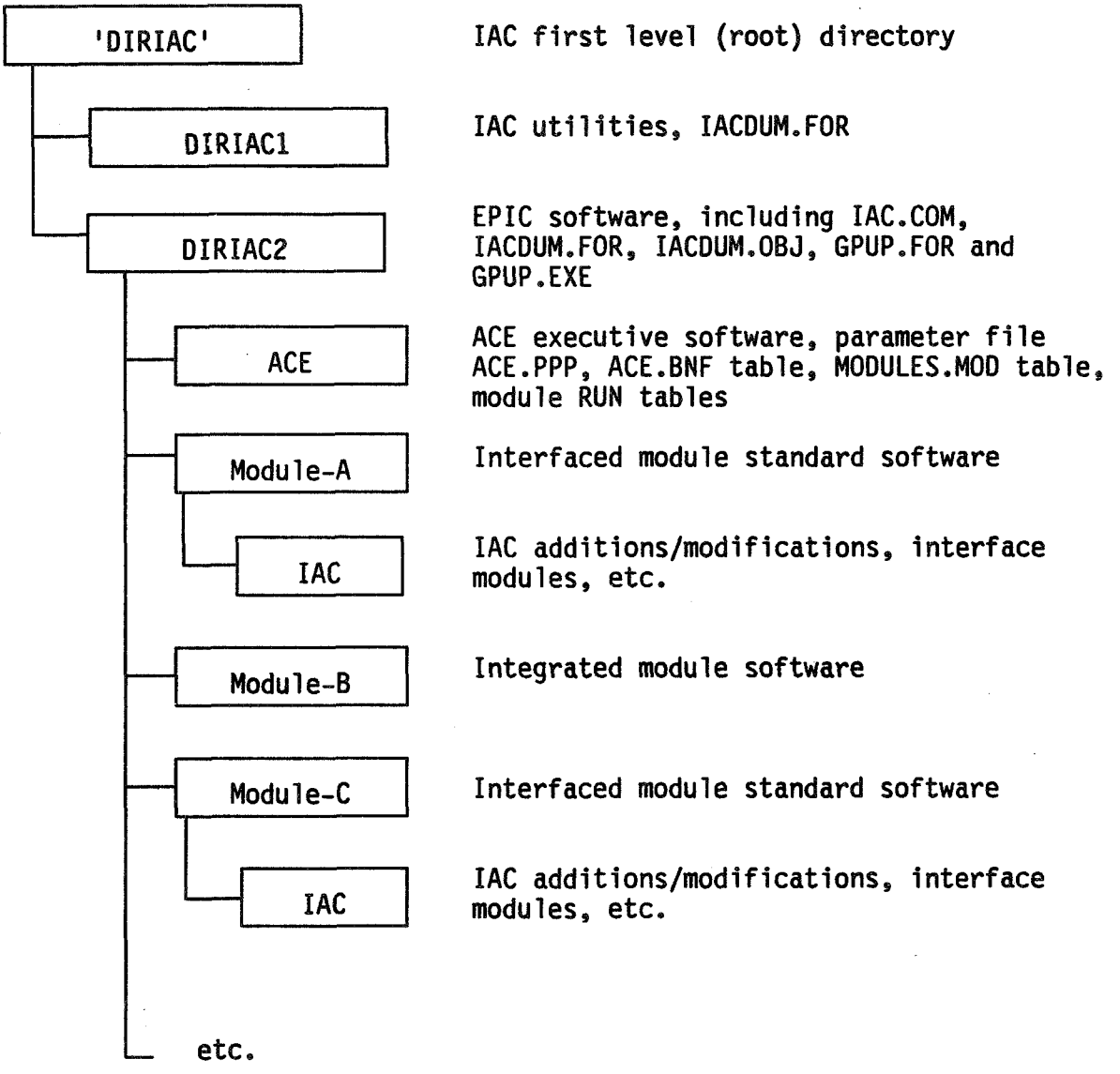
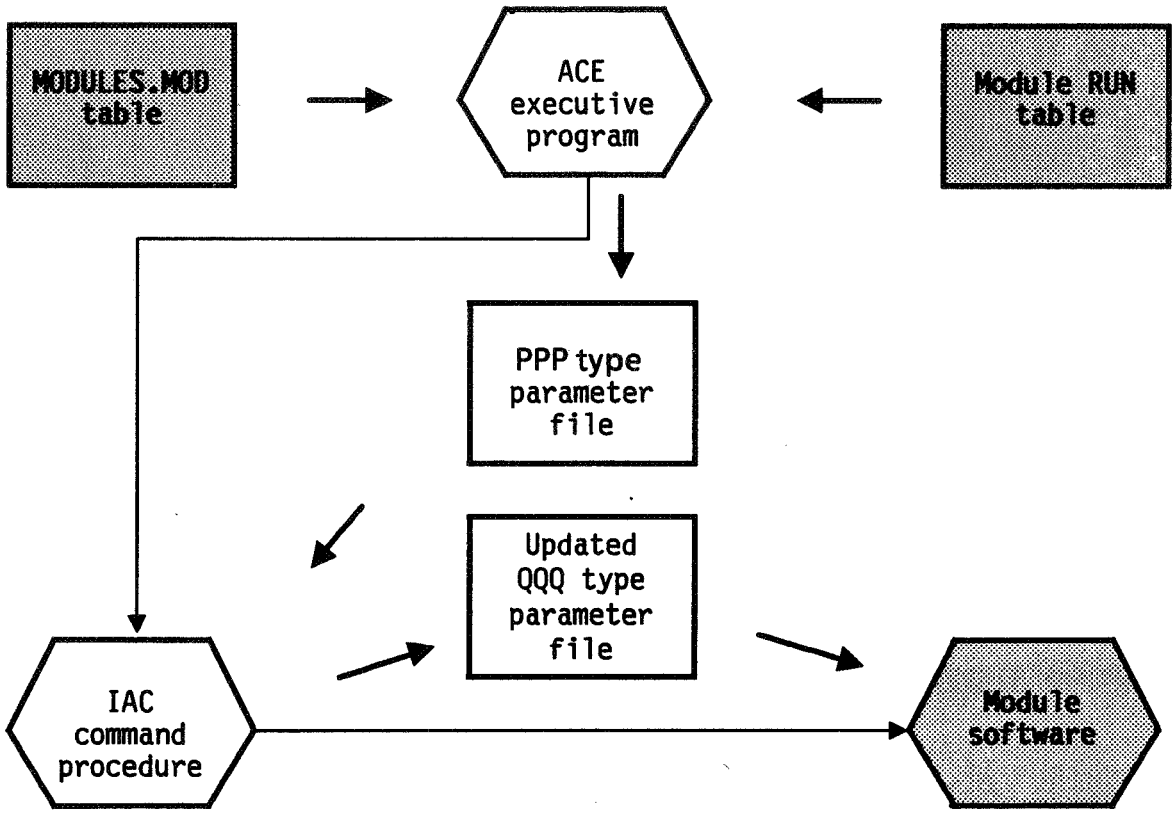


Figure 7.0-1: Typical IAC Directory Organization



Note - Shaded boxes must be supplied by the module developer.

Figure 7.0-2: Module Execution Schematic

Name	Description
RUNID	Module unique RUN identifier (maximum 8 characters)
USERMODE	INTERACTIVE or BATCH
DIRDEF	User default directory
DIRSCRATCH	Installation dependent scratch directory
DIRDB	Spec of currently open database
USERNAMEDB	Database username string (maximum 10 characters)
IACGSN	Name of the IAC global symbol (e.g. IAC or IAC2)
DIRIAC	Root portion of IAC directory spec
PPPSTATUS	Parameter file status (KEEP or DELETE)
QQQSTATUS	Updated parameter file status (KEEP or DELETE)

Table 7.0-1 GEN Type Parameters

IAC Directory Organization - At each IAC installation site, the IAC system support software and data will reside in certain defined directories and subdirectories. Included here are the IAC utilitites, the EPIC (Enhanced Program Interface Capability) software tools, and software associated with many of the IAC system modules. A typical IAC directory organization is depicted in Figure 7.0-1. Access to the software at a particular installation is facilitated by the parameter named DIRIAC (see also the Table 7.0-1 and the Parameter File topic). DIRIAC defines the root portion of the IAC directory spec, down through the first (top) level directory name. For example, if the IAC system at that installation is stored on disk "DB1" under the first level directory name "OURIAC", the value of the DIRIAC parameter would be "DB1:[OURIAC".

The two second level IAC directories have hard-defined names as follows.

Name	Description
DIRIAC1	Storage for IAC utilities and dummy routines.
DIRIAC2	Storage for certain EPIC software applicable to more than one module. Also serves as parent for some of the IAC module subdirectories (generally one or two subdirectories per module).

For example, the complete directory spec for the IAC utilities would be given by "'DIRIAC'.DIRIAC1]". As another example, if IAC software for a particular module were stored in subdirectory "XYZ", the complete directory spec for that software would be given by "'DIRIAC'.DIRIAC2.XYZ]".

IAC Global Symbol - The module implementor should have a basic understanding of the definition and use of the IAC global symbol. This symbol is used to invoke IAC, either initially by the interactive terminal user, or subsequentially via an interactive or batch spawning process.

The symbol has a name and associated value, and will usually have been defined by the host system manager. It might in fact be convenient to have two alternate symbols, for example, one having the name IAC and used for the production environment, and the other having the name IAC2 and used for testing and for a new software release.

The global symbol value consists of the following parts.

a) @IAC-command-procedure-spec
b) IAC-global-symbol-name
c) DIRIAC-root-directory-spec

Part (a) executes the IAC command procedure (see next topic). This procedure is the starting point when initiating the ACE executive or any other module within the IAC system. Parts (b) and (c) are passed to the IAC command procedure as procedure parameters. Part (b) is the global symbol name. This name is included within the global symbol value so that the name may be saved (see IACGSN parameter, Table 7.0-1) for possible downstream use. Part (c) is the root portion of the IAC directory spec, as mentioned in the previous topic. This root is used to locate IAC system software and data.

IAC invocation takes the form

```
IAC-global-symbol-name  IAC-parm-file-spec
```

Here the IAC-parm-file-spec is optional, with the default being the spec of the standard parameter file for the ACE executive. This spec is passed to the IAC command procedure, in the same manner as are parts (b) and (c) above.

As an example, suppose that the global symbol name has been defined as

```
IAC2
```

and the corresponding value as

```
@DB1:[DIRX.DIRIAC2]IAC2.COM·IAC2 DB1:[DIRX
```

Then the following invocations might be employed.

1) IAC2
2) IAC2 TEST.PPP

The first invocation would initiate the ACE executive module. The second would directly initiate the module referenced by the TEST.PPP parameter file, without going through the executive.

Module Execution Sequence - The module execution schematic is shown in Figure 7.0-2. The light line arrows in the figure indicate temporary transfer of control between programs and/or command procedures, while the heavy arrows indicate writing (creation) or reading of a file by a program or procedure. The module developer need be concerned only with providing the module software and RUN table, and adding one row to the MODULES.MOD table.

The parameter file provides a basic interface which connects the executive program with the module. It contains keyword parameter definitions in card image form. A parameter file is usually generated by the executive, but it could be created by other means for independent checkout or usage of a module. Definition and use of the parameter file is discussed more fully in the next topic.

Referring still to Figure 7.0-2, the executive program performs the following sequence of tasks for executing a module.

1) Process the user's RUN command for the module, including any keywords supplied for parameter definitions.
2) Create a RUNID consisting of an 8-digit RUN identifier unique to this invocation (based on the host system clock).
3) Create the parameter file containing GEN, MOD and RUN type parameters (in that order) and store this file in the user directory. The GEN type parameters are general parameters, e.g. RUNID, which may have potential significance for all modules; the current list of these parameters is given in Table 7.0-1 (destination is ANY). The MOD and RUN type parameters are created by the executive using the MODULES.MOD table and the module RUN table, respectively, shown in Figure 7.0-2. The MODULES.MOD table pertains to all IAC modules and contains one row for each module. The RUN table is specific to a particular module and contains one row for each parameter available via the executive RUN command for that module. These tables are described more fully in the later topics.
4) Spawn a subprocess to execute the IAC command procedure shown in the figure. The argument to the host spawn utility is the FORTRAN string IACGSN//' I'//RUNID//'.PPP' (note that the value of IACGSN is the

available system global symbol name). The IAC procedure is constant and is stored as part of the IAC system software. The parm-file-spec is passed to the IAC procedure, in order to make the parameter file accessible to the module associated software; the spec is of the form Irunid.PPP, where runid is the value of the RUN identifier RUNID.

The IAC procedure executed in task (4) above performs the following tasks.

1) Define the IAC global symbol name IACGSN and the root directory spec DIRIAC as local symbols (insert definitions into the host local symbol table, using the host parameter subvalues contained within the value of the IAC global symbol).

2) Define PARM_FILE_SPEC as a local symbol equal to the passed parameter parm-file-spec (default is "'DIRIAC'.DIRIAC2.ACE]ACE.PPP").

3) Run GPUP.EXE to read the PPP parameter file, generate any missing GEN type parameters, define as a local symbol each parameter whose desination is ANY or SYM, and write the updated QQQ parameter file.

   For missing GEN parameters, default values are obtained as follows.

   RUNID using the host system clock
   USERMODE from the host
   DIRDEF from the host
   DIRSCRATCH equal to DIRDEF
   DIRDB equal to blank
   USERNAMEDB equal to blank
   IACGSN from second part of IAC global symbol value
   DIRIAC from third part of IAC global symbol value
   PPPSTATUS equal to KEEP
   QQQSTATUS equal to DELETE

   The spec of the updated parameter file is of the form Irunid.QQQ, where runid is the value of the RUN identifier RUNID.

4) Delete the PPP parameter file, if the value of PPPSTATUS is DELETE.

5) Use value of MODNAME to display a message indicating begin of module. Display value of RUNID.

6) Define PARM_FILE_SPEC as a local symbol equal to Irunid.QQQ.

7) Execute the module software, via the host command "modjcl", where modjcl is the value of the MODJCL attribute in the MODULES.MOD table. This command initiates execution of the module software, e.g. module command procedure(s), executable program(s), etc. Although the module software typically consists of a command procedure which runs an executable module, this software is arbitrary and the access to it is defined by a module's implementor via the MODJCL attribute.

8) Use value of MODNAME to display a message indicating end of module.

9) Delete the updated QQQ parameter file, if the value of QQQSTATUS is DELETE.

Additional EPIC Software - In addition to the general module execution software just discussed, IAC provides the module implementor with two high-level generic procedures, EPICA.COM and CEPIC.EXE. These procedures are stored in the DIRIAC2 directory.

EPICA.COM may be identified as the starting point for module software execution (see MODJCL attribute in the MODULES.MOD table). EPICA is a command procedure which depends on two local symbols, EXE and JCL. If EXE is not null, then EPICA passes control to the executable file whose spec is the value of EXE. If EXE is null, then EPICA passes control to the command file whose spec is the value of JCL. In either case, EPICA provides certain execution enhancements (e.g. a graceful return of control, in case the module software encounters host system identified errors or the user instigates a host system provided abort).

CEPIC.EXE may be used to retrieve parameters of desired types and destinations from the parameter file, and to define them as local symbols. CEPIC assumes that the parameter file spec is defined by the local symbol PARM_FILE_SPEC. The CEPIC procedure also expects one host procedure argument to be passed to it, of the form tdlist/units. The tdlist string is a list of type.destination identifiers (same form as for the utility IACPFR, documented in Section 6). The /units string is an optional list of two FORTRAN unit numbers separated by a comma (default string is "/30,6"); the first unit is used to read or write the parameter file, and the second is used for any messages which might be produced.

Parameter File - The parameter file is an IAC standard interface for passing parameters between different software procedures and programs. It contains keyword parameter definitions in 80-character card image form. Each record (card image) has the following organization.

Item	Characters	Format
name	1:10	A10
type	11:15	A5
destination	16:20	A5
not used	21:28	8X
length	29:30	I2
value	31:80	A50

A-format items are left-justified. The name, type and destination items are alphanumerics (any blanks must be trailing), with first character alpha. Alpha includes upper-case A-Z and underscore "_".

Name is the parameter identifier.

Type indicates the origin of the parameter and may be one of the following.

GEN = general parameter
MOD = MODULES.MOD table parameter
RUN = RUN table parameter

Destination may be one of the following.

ANY = any destination
EXE = module executable code
JCL = module job control language (command) procedure
SYM = host symbol definition table

Note that if a particular parameter name occurs more than once on the parameter file, then a special convention is followed by the EPIC modules GPUP.EXE and/or CEPIC.EXE (see the Module Execution and Additional EPIC Software topics) if they insert any of the associated values into the host

symbol table. The convention is that values associated with the same type and destination are sequentially catenated using a comma delimiter; a value associated with a unique type and destination is not catenated. If, after catenation, more than one parameter of that name exists, the last parameter defines the value in the host symbol table.

The length indicates length of the value item on the current record (value characters start in column 31). A value may be continued onto additional records as desired, with each record containing at most 50 characters of the value; each record after the first contains the name CONTINUE. Note that unused columns following the value item may be used for comments, etc. Selected records from a parameter file may be read by an executable module into a module supplied array, via a call to the utility IACPFR; see Section 6.4. The parameter file may also be processed directly by any of the module associated software.

MODULES.MOD Table - The attributes defined in the MODULES.MOD table are the following.

Name	Type	Description
MODNAME	C10	(Module name)
MODALIAS	C10	(Module alias)
RUNTABLE	C*	(spec of module RUN table)
MODJCL	C*	(Module JCL execution command)
PARNAMES	[*] C10	(Parameter names)
PARTO	[*] C5	(Parameter destinations)
PARVALUES	[*] C*	(Parameter values)
IACNAMES	[*] C10	(Names for use by IAC system)
IACVALUES	[*] C*	(Values for use by IAC system)
HELP	[*] C*	(Module associated help information)
EXAMPLES	[*] C*	(Module associated examples information)

Character values are left justified.

Values for the MODNAME and MODALIAS attributes are 10-character alphanumerics (which may include trailing blanks), with the first character alpha. Alpha includes upper-case A-Z and underscore "_". These parameters are written onto the parameter file with a destination of ANY.

The RUNTABLE value is the spec of the RUN table relation for the module; the type in this spec should be RUN, as in "NASTRAN.RUN". The total spec is formed by catenating this spec onto "'DIRIAC'.DIRIAC2.ACE]".

MODJCL is a string, e.g. "@X.COM". When processed as a command by the host operating system, this string causes execution of the module software. The MODJCL value may contain any valid host command syntax, e.g. global symbol references. This parameter is written onto the parameter file with a destination of ANY.

vectors have the same length, equal to the number of parameters. (If not needed, a single blank character may be used for each.) These parameters are written onto the parameter file with destinations equal to the respective PARTO values.

The IACNAMES and IACVALUES vectors may give names and associated values, respectively, of items to be used by the IAC system. Both vectors have the same length, equal to the number of items. (If not needed, a single blank character may be used for each.) Currently these attributes are used to define information for the ACE LINK command; the names LINKMAIN, LINKOBJ, LINKLIB and associated values are used to define specs for the respective main program, object file(s) and library file(s) to be used in linking a user version of the module (LINKOBJ and LINKLIB may occur more than once).

The HELP and EXAMPLES vectors define help and examples lines, respectively, associated with the overall module; e.g. brief description of module, information on consultation support, etc. This information may be displayed at the request of a user during input of an ACE executive RUN command.

Module RUN Table - The attributes defined in a module RUN table are the following.

Name	Type	Description
IACKEYNAME	C10	Parameter name used in ACE RUN command
MODKEYNAME	C10	Parameter name used by module
IACTOMOD	C10	ONETOONE or MANYTOMANY
TYPE	C10	Value type, e.g. I1, Cn, C*, HFILE
TO	C10	JCL, EXE, SYM or ANY
MODREQ	C10	OPTIONAL, DEFAULT or NODEFAULT
VALRANGE	C10	FIX or NOFIX
LINKPAR	C10	blank or name of related parameter
IACOP	C10	IACOP or NOIACOP
MODOP	C10	MODOP or NOMODOP
LISTS	[*] C*	(name1 list1 $ name2 list2 $ etc.)
VALNAME	C*	Prompt name
HELP	[*] C*	(Help information)
EXAMPLES	[*] C*	(Examples information)

Values of the IACKEYNAME and MODKEYNAME attributes are the names used for referencing the parameter within the executive and the module, respectively. These values are 10-character alphanumerics (which may include trailing blanks), with the first character alpha.

IACTOMOD defines the parameter multiplicity from the executive to the module. MANYTOMANY is used If and only if the parameter is a list. (see Note 10 in Section 2.1); this results in creation of a class GOR node (see Table 2.1-1) within the command language.

TYPE is the data type of the parameter, e.g. R1 or C* (see Table 2.1-1).

The TO attribute defines the destination for the parameter; JCL indicates module job control procedure, EXE indicates executable module, SYM indicates insertion into the host symbol table prior to execution of the module software, and ANY indicates multiple or arbitrary destinations. The module job control procedure may access parameters directly, or by executing the

MODREQ defines the module requirement for the parameter to be present on the parameter file; OPTIONAL indicates that it need not be present; DEFAULT indicates that it must be present but that the LISTS attribute defines a default parameter value for use when a value is not provided in the RUN command; and NODEFAULT indicates that it must be present and must be provided by the RUN command.

The VALRANGE attribute has the value FIX to indicate that only certain parameter values are permitted in which case the range of values is defined by the LISTS attribute; the value NOFIX indicates that any values are permitted.

LINKPAR may give the name of another parameter in the RUN table to which this parameter is related in some way; this attribute is provided to allow for possible future extensions.

The IACOP attribute has the value IACOP to indicate that the parameter values may contain IAC defined operators, in which case the list of valid operators is given by the LISTS attribute; the value NOIACOP indicates that no IAC defined operators are to be used. The MODOP attribute is defined in a similar manner, and indicates whether the parameter values may contain module defined operators.

The LISTS vector defines lists of items to be used in conjunction with other attributes for a parameter. Each list begins with the name of the associated attribute, followed by the list of values, and ends with a dollar sign terminator "$". An actual "$" value may be represented by two successive "$" values.

VALNAME is the prompt name for the parameter. Note that if IACTOMOD has the value MANYTOMANY, the VALNAME value must begin with the 5 characters "list(".

The HELP and EXAMPLES vectors define help and examples lines, respectively, available to the user during processing of the executive RUN command.

8.0 REFERENCES

1. "IAC Executive Summary," W. J. Walker, R. G. Vos, G. A. Price and E. W. Brogren, NASA CR-175196, Boeing Aerospace Company, February 1984.

2. "Integrated Analysis Capability for Design of Large Space Systems," Presentation by W. J. Walker, 21st AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, May 1980, Seattle, WA.

3. "Integrated Analysis Capability," J. P. Young, R. G. Vos, G. K. Jones and H. P. Frisch, 2nd Chautauqua on Productivity in Engineering and Design: The CAD Revolution, November 1982, Kiawah Island, SC.

4. "Development and Use of an Integrated Analysis Capability," R. G. Vos, W. J. Walker, D. L. Beste, G. A. Price, J. P. Young and H. P. Frisch, 24th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, May 1983, Lake Tahoe, NV.

5. "Design of Multivariable Controllers Using the Integrated Analysis Capability (IAC)," J. A. Bossi, G. A. Price and S. A. Winkleblack, Workshop on Identification and Control of Flexible Space Structures, June 1984, San Diego, CA.

6. "Flexible Spacecraft Controller Design Using the Integrated Analysis Capability," J. A. Bossi, G. A. Price and S. A. Winkleblack, AIAA Guidance and Control Conference, August 1984, Seattle, WA.

7. "MSC NASTRAN Version 61," Theoretical/User/Programmer/Application Manuals, MacNeal-Schwendler Corporation, 7442 N. Figueroa Street, Los Angeles, California.

8. "A Digital Computer Program for the Dynamic Interaction Simulation of Controls and Structure (DISCOS)," C. S. Bodley, A. D. Devers, A. C. Park, and H. P. Frisch, NASA Technical Paper 1219, Volumes I-III, May 1978.

9. "TRASPLOT User's and Programmer's Guide," NASA/GSFC Contract NAS5-24300, Computer Sciences Corporation, CSC/TM-80/6177, July 1980.

10. "Guide for Use of the Thermal Radiation Analysis System II (TRASYS II) on the VAX 11/780 Computer," NASA/GSFC Contract NAS5-24300, CSC/TM-80/6166, September 1980.

11. "TRASYS II Test Cases, VAX Version 1.4," NASA/GSFC Contract NAS5-26299, Computer Sciences Corporation, CSC/TM-81/6082, May 1981.

12. "SINDA User's Manual with SINFLO Addition," Informal documentation revision directed by NASA/GSFC, Sperry Support Services, 1980.

13. "ORACLS - A System for Linear-Quadratic-Gaussian Control Law Design," E. S. Armstrong, NASA Technical Paper 1106, April 1978.

14. "ORACLS - A Design System for Multi-Variable Control," E. S. Armstrong, Dekker Publishing, 1980.

15. "LINPACK User's Guide," J. J. Dongarra, J. R. Bunch, C. B. Moler and G. W. Stewart, SIAM, 1979.

16. "Matrix Eigensystem Routines - EISPACK Guide," B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klena and C. B. Moler, Volume 6, Second Edition, Springer-Verlag, 1976.

17. "Theoretical and Programming Manual for the MODEL Simulation Language Translator," B. G. Zimmerman, June 1982, NASA Goddard Space Flight Center, Greenbelt, MD.

18. "RIM - Relational Information Management System, Version 5.0 User Guide," Boeing Computer Services Company, January 1982.

19. "Modern Numerical Methods for Classical Sampled Systems Analysis, (SAMSAN Version 2) User's Guide," H. P. Frisch and F. H. Bauer, COSMIC Program Number GSC-12827, January 1984.

APPENDIX A - NEW USER'S INTRODUCTION TO ACE

So you've been told to use the I-AC to run your study, get part of the results and put them into the plot package. Despite assurances that it is "user-friendly" you want to know a little more about it before jumping in. And here you are. In this appendix we are going to try to make you comfortable using ACE, the IAC program that accepts and acts on commands.

A.1 HOW ACE WORKS

Each installation has its own method to start up ACE with some combination of JCL, symbols and special commands. Once it begins, it identifies itself on your terminal. From then until you end it, it works "interactively" with you: it prompts you for some input, you reply in some way, and it goes off to examine what you gave it. The next prompt depends on the results of that examination. If it thinks your reply was in error it gives you a message and prompts you again. If it was acceptable then you get a prompt for the next portion of the command, if any more is required. If ACE doesn't think you are through it will use a prompt to ask for more.

How can you respond to a prompt? You can type in something to "answer" or "fulfill" the prompt; you can hit a carriage return to reject the prompt (but sometimes ACE won't let you do that); or you can give control commands consisting of a % followed by one or more letters to get explanations or examples for this prompt, or move to some other point in the command "tree" you have formed so you can get help or begin anew from that place. These control commands are briefly explained in Section A.4. If you are typing in text you may enter part of the full reply to the prompt, all of it, or all of it plus some or all of the rest of the command. The ACE > prompt is looking for a full command. A prompt, by the way, consists of everything to the left of the > while what follows to its right (if anything) is called the "autoinput". Autoinput is the first portion of an acceptable response that you would have to enter anyway; ACE just does it for you. If you reject the prompt, it throws away the autoinput too.

As you are entering commands and waiting for the next prompt or your output, it may occur to you to ask, "What in the world is ACE doing?" When you are

writing everything correctly and putting in the complete command each time, the time can drag. However, when you make a mistake, or don't remember what comes next, the time is well spent because ACE is doing a lot of work to help you out in such cases. It lets you pick up from where you made an error rather than retype the whole command. If you stop short on the command it tells you what it wants next, and you can get it to display all of the possiblities at that point. When you use control commands it allows you to review what you have already done (according to the command structure -- see the diagrams in Section 2), start over at some point therein, or even kill the whole command. A very useful control letter is H: enter **%H** to get Help for the current prompt.

One of the reasons for using IAC and its ACE program is to access a command store of data files. The files are kept on the host computer as standard files, but they are also known to a special catalog that ACE can access. The catalog contains the host file name, an IAC name for the file, and possibly a title and some key words that describe the file, for each file it catalogs. It and the files it catalogs are called a "database". Many of these files will be "structured" which means they can be put into an IAC workspace such as the one ACE uses. Once in the workspace you can do all sorts of interesting things to the file: edit or display it, "Query" it (look for particular values in certain places) or save the new version of it. To do some of these you must **OPEN** the database, which means access the catalog file; you can **GET** the files of interest into the workspace; and you can **PUT** new files or versions back into permanent storage in the host computer system. A word of caution: if you forget to **PUT** new files into the host system before you **EXIT**...they are gone. To find what files are in an open database you can use the LIST command with the qualifier **/DATABASE**. To see the structure of a file in the workspace, use the **SHOW FILE** command.

There are other ACE commands to print or plot your data, run analysis programs (called "modules") or perform miscellaneous session-control operations. The commands you will probably use most frequently are demonstrated in Section A.3. One timesaver you may want to use while entering any command is abbreviation. You may abbreviate any word except for

a file name down to the inital letters that make it distinct from any other acceptable word. If ACE is looking for one of IO, MACRO, MESSAGE, or UCATT (in the SET command) you could type in I, MA, ME, U or any longer portion of these words. Special words beginning with a $ but not ending with a number may also be abbreviated. "Attribute" names may be abbreviated.

Another ability of ACE which can save you time is its memory of your most recent entry to certain prompts. When you wish to answer that prompt in the same way you did last time, you type in $S. The $S can be used at the corresponding place in the middle or end of a line you are typing. However, it can only be used at those places in commands whose diagrams in Section 2 show a box with $S in it as an alternative. For example,if your last command was PRINT ARM.STRESSES NODE,X,Y NODE LE 40 then you could repeat it by typing PRINT $S $S $S, the first $S corresponding to ARM.STRESSES, the second to NODE,X,Y and the last to the condition, NODE LE 40.

A.2 ACE TERMINOLOGY

We have tried to use the most concise and descriptive words we could find for ACE prompts and the terminology used in the documentation. Occasionally a term may be a little too concise, and you won't know what it means until you've had a chance to use it...and find out what it means. In this section we try to alleviate that problem somewhat. Starting off are explanations of common terms encountered in reference to the database. Then come elaborations of a few of the more notorious prompts. Finally there is a miscellany of words that don't fit either of the above categories.

In the database and workspace we refer to arrays, relations, attributes and tuples. If you have a good grounding in relational database theory, the last three terms are familiar to you. The first one, **array**, is not a synonym for "matrix" within IAC. Instead it is a composite of several relations which are all bound together in a special way. Let's initially take the term relation. It is a data structure whose data obey a fixed **relation**. Well, that's a general and concise way of stating it but it doesn't say very much. Picture it as a structure of rows and columns. The relation fixes the number of columns, and gives each column a descriptive name (TIME,X,Y,STRESS); the column is called an **attribute** of the relation. Associated data values are

kept in the same row (at TIME 2.1, X was 1.03, Y was -2.25 and STRESS was .0025); each row is called a **tuple** of the relation and there can be an arbitrary number of them. Each attribute (column) has a fixed type (integer; real or complex, with normal or double precision; logical; or fixed or variable length character) and is either scalar or nonscalar. Nonscalars have a fixed number of dimensions and may have the size of each dimension the same for all tuples (rows) or possibly different in each tuple.

IAC arrays are more general. They have a "core" matrix of data attribute(s); these attributes may not contain variable lengths or dimension sizes (this restricts only the core matrix). The core matrix has one or more dimensions. Each dimension may have associated with it an IAC relation which has the same number of tuples as the size of the dimension; each of the relation's tuples is linked to a value within the dimension, in effect providing labels for it. If the dimension is the first dimension of the matrix, then the associated relation is said to belong to **index** one. The core matrix belongs to **index** zero. Likewise, other relations belong to the index numbered with the number of the dimension that they "label". An example is given in Section 5.3.1.

IAC files have a name known to the catalog (and hopefully to you) that may differ from the naming conventions of the host file system. It consists of a one-to-ten character word (with "_" allowed), an optional number, another one-to-ten character word, and an optional version number. This gives you considerable latitude in naming files descriptively. An example of the full format is BASE_CASE:14.LONG_ARM;5. Names of files are called **ra-specs**, for relation/array specifications. When the file is known to be a relation, ACE prompts for a **rel-spec**, and when it expects an array it prompts for an **arr-spec**.

Frequently ACE expects a list of one or more things. Lists can be entered in one of two ways, whichever you find convenient. You can enclose the whole list inside a pair of parentheses, in which case commas may or may not be given between the list items. Leave off the parentheses if you prefer and put a comma between each item in the list. Some items themselves are compound (or even other lists!); the comma is required only at the finish of the whole item. When entering print-names, for example, you can say

NODE,X,Y,STRESS or (NODE X,Y STRESS). The comma in the parenthesized list was optional.

The analysis programs within the IAC system are called **modules**. ACE is also an IAC module. You can use any module within ACE by typing **RUN**, followed by the module-name -- including ACE, which can be handy if you need another version of the ACE workspace.

The terminal is not the only way to get commands or parts of them to ACE. Macros and amacros, described in Section 2.2, allow you to reference previously written text (macros) or to send a prompt to the terminal from a command file (amacros). A command file can be used with the SET IO INPUT=hostfile command. The command file functions as if you yourself were typing in exactly what's in the file; in fact, you will see the prompts, acknowledgements, output and error messages on your terminal from all of the commands. One special requirement of the file is that its last command be either CLEAR IO or SET IO INPUT='*'. Otherwise ACE will abort and you will lose your workspace contents.

A.3 WHAT FOLLOWS

There are three more sections to this appendix. Section A.4 contains a terminal session displaying examples of many frequently performed ACE activities. Section A.5 contains a list of the control commands and some explanation of how they function. Finally, Section A.6 is a terminal listing of the Help and Examples of each ACE command at its highest level (the top of the diagram of the command in Section 2).

The terminal session in A.4 contains ACE comments in which we try to describe what's going on and why. You can put comments in your ACE commands too by putting them between exclamation points. If the comment is at the end of a line, only the leading ! is required. These are useful if you are recording your session with a local printer or the ACE ECHO file, or setting up a command file.

A.4 A PAINLESS INTRODUCTION TO ACE COMMANDS

```
BEGIN ACE RUNID=07405643
Enter command, or EXIT to terminate, or %H for help.
ACE >%HE
Your response to any prompt may be all or part of a remaining command.
A command consists of a statement and an optional disposition.
Your response may also be a sequence of control characters, preceded
by "%".  The following are control characters.
  "H" to obtain help for current node.
  "E" to obtain examples for current node.
  "P" to escape backward to previous command token entered.
  "K" to kill the whole command, deleting all its tokens.
There are other control characters for special functions.  To see a
list of them, enter "%Z".  A special mode can be entered, called
EXPLORATORY mode, in which the BNF syntax tree can be explored from
the current position.  The regular mode is called INPUT mode, which
accepts ACE command tokens.  Exploratory mode never affects the
current command.
Multiple control characters may be catenated, e.g.
  "%DHE" to obtain next-level node prompt, help, and examples.
At some prompts, you may enter one of the following.
  <CR>, i.e. carriage return, to reject current node prompt.
  "$S" to indicate command input same as last input of that type.
You may now enter all or part of a command, or a control sequence
such as "%DDH".
%DDH
OPEN
GET TRUSS.NODES, *.SAVE
ACE >OPEN      !   To get at your IAC files in ACE, you have to OPEN their     !-
               !   database.  In this example, an existing database is opened. !+
               !   Instead of typing the whole command at once, we'll just    !+
               !   start with the command word and let ACE tell us what else  !-
               !   is needed.                                                 !
list(OPEN-parms) >(%H
A list of keyname parameters used to open an old (existing) or new
database.  The following four OPEN parms are available.
  DIRECTORY = directory
  USERNAME = username
  STATUS = status (default status is OLD)
  ACCESS = access (default access is CONCURRENT)
list(OPEN-parms) >(DIRECTORY='[JG]'     !   The HELP we requested tells us    !+
                                        !   what our options are.             !
, >,USERNAME=JG     !  This "name" is a word which identifies us in this      !+
                    !  OPEN of the database as either the owner or nonowner   !-
                    !  of the files.  Protection can be set on each file to   !+
                    !  distinguish the two.  The name of the owner is given   !+
                    !  when the catalog is created.                           !
```

```
, >,
STATUS=status >S=
ACCESS=access >A=
) >)      !   Notice how we rejected the prompts for comma, STATUS and ACCESS  !+
          !   by hitting a carriage return right away.  The autoinput will be  !+
          !   accepted if we type a space first.  We are accepting this auto-  !+
          !   input of ")" by typing a space (as well as this comment) before  !+
          !   hitting the return key.  The command is now complete and will    !+
          !   be executed.  ACE will tell us of any errors that cause the      !+
          !   command to fail.                                                 !
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!    When you want to create your own database, you OPEN it with STATUS  !+
     !    = NEW.  Only one database may be open at a time, with any old one   !+
     !    being closed first.  Only one database may be present in a given    !+
     !    directory (because the catalog file name is fixed).                 !+
     OPEN (STATUS=NEW, DIRECTORY=[JG.DIRIAC2], USER = FROG)  !  for example. !
*ERROR* The token at the end of:
        OPEN ( STATUS = NEW , DIRECTORY = [
        did not satisfy any of the following prompts:
 directory
directory >'[JG.DIRIAC2]'  USER=FROG )    !   Ooops.  Well, ACE is forgiving  !+
                                          !   of such things.  The directory  !+
                                          !   was supposed to be a single C*  !+
                                          !   "token" but since it had brac-  !+
                                          !   kets, it had to be a "string"   !+
                                          !   which means enclosed in '...'.  !
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >OPEN U=JG,D='[JG]'     !   This shows the other form of the "list."      !
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >LIST/D *.*       !   The LIST command gives you the names of the files in  !+
                      !   the directory.  A list of names, possibly with the    !+
                      !   "wild card" asterisk, is looked up.  Here the list    !+
                      !   consists only of *.* which means "all files."  Note   !+
                      !   that the database is searched because of the quali-   !+
                      !   fier /D.  Without this qualifier, the workspace dir-  !+
                      !   ectory would be searched;  currently there are no     !+
                      !   files present in our workspace.  That's next.         !
SPEC                                      CLASS
MESH:1.A;1                                RELATION
MESH:1.B;1                                RELATION
ST:1.MACROS;2                             RELATION
Z:1.REL;3                                 RELATION
LARGE:1.ARRAY;1                           ARRAY
A:1.ARRAY;1                               ARRAY
TEST:1.REL;3                              RELATION
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!    SPEC and CLASS are titles in the above list.  Next we will retrieve !+
     !    some of these files, making them available in the workspace for     !+
     !    further activities.                                                 !+
     GET/VERIFY
/data >/
/text >/
file-specs >MESH.*,TEST.REL     !   This could be a list (as done here) or    !+
                                !   $S to indicate "the same as last time"    !+
                                !   we gave file-specs.  There has not been   !+
                                !   a "last time" yet, during this session.   !
```

```
*INFORMATION* GET OF DATABASE MESH:1.A;1
              TO WORKSPACE MESH:1.A;1 COMPLETED
*INFORMATION* GET OF DATABASE MESH:1.B;1
              TO WORKSPACE MESH:1.B;1 COMPLETED
*INFORMATION* GET OF DATABASE TEST:1.REL;3
              TO WORKSPACE TEST:1.REL;3 COMPLETED
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >BEGIN   OPEN U=FROG,D='[JG.DIRIAC2]'   PUT/V $S   END                    +
     !   This is a compound statement.  We wanted to copy files from one of  !+
     !   our databases into another, so it was necessary to GET them into    !+
     !   the workspace, OPEN the target database, and PUT them out to it.    !+
     !   Note the free use of abbreviations.  BEGIN could have been B and    !+
     !   OPEN just O, but PUT had to be spelled out so it wouldn't be con-   !+
     !   fused with PUTS and END needed at least EN to distinguish it from   !+
     !   EXIT.  The extra spaces between commands are optional.              !
*INFORMATION* PUT WORKSPACE MESH:1.A;1
              TO DATABASE MESH:1.A;1 COMPLETED
*INFORMATION* PUT WORKSPACE MESH:1.B;1
              TO DATABASE MESH:1.B;1 COMPLETED
*INFORMATION* PUT WORKSPACE TEST:1.REL;3
              TO DATABASE TEST:1.REL;3 COMPLETED
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   Normally the datastructures you will use would be generated by a    !+
     !   module such as NASTRAN ("datastructures" is a fancy word for files  !+
     !   but implies there is some organization within each file).  Some-    !+
     !   times you will need to create, load and alter the information in a  !+
     !   datastructure yourself.  ACE allows you to do these interactively.  !+
     !   First you DEFINE the file as one of two structure classes:          !+
    DEFINE RELATION NEW.RELATION
rel-att-defs >%H
New relation attribute definitions, consisting of one of the following.
  list(att-defs) = list of attribute definitions (defines new attributes)
  ra-spec/arr-index = reference to an existing relation or array
                      (clones attributes)
rel-att-defs >%E
NODE,I1/I5, X,R1/F10.3, Y,R1/F10.3
BMESH.NODE
A.TEMP/1
rel-att-defs >(NAME,C*/'1X,A'     !   The stuff after the slash is the for-   !+
                                  !   mat that would be used if the PRINT or  !+
                                  !   similar command is used on "NAME."  It  !+
                                  !   is optional; there are defaults.  The   !+
                                  !   comma  between the name and type is     !+
                                  !   also optional.  C* means variable size  !+
                                  !   character string.                       !
, >,EXT I1     !   The phone extension can be kept in an integer and use the  !+
               !   default integer format (which is 1X,I10).                  !
/att-format >/
, >,COMMITTEE[6] L1)     !  This attribute would have logical (TRUE/FALSE)    !+
                         !  values for committee assignments.  The six com-   !+
                         !  mittee names are not kept in this relation.       !
```

```
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >SHOW FILE NEW.RELATION     !   To see what we've created...                !
DATASTRUCTURE=NEW:1.RELATION;1, CLASS=RELATION
TUPLES      NAME         DIMENSIONS TYPE  FORMAT
          0 NAME                    C*    1X,A
            EXT                     I1    1X,I10
            COMMITTEE  6            L1    1X,L10
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   Now let's load this relation with some values.  One VERY useful     !+
     !   option is to ask for notice of where we are in the loading process  !+
     !   (which "tuple" or row and what "attribute" or value should be next  !+
     !   or was last entered).                                               !+
    LOAD /NEXT NEW.RELATION   'SNORKLEWHACKER, John' 5335 (T,F,F,F,F,F)
Next LOAD = tuple 2 NAME
, >,'GRINSCH, Zephus' 2620 (F,F,T,T,     !   Note that dimensioned values of  !+
                                         !   fixed size must be enclosed in   !+
                                         !   parentheses.  There are special  !+
                                         !   rules for variable-sized dimen-  !+
                                         !   sions when you load them.        !
Next LOAD = tuple 2 COMMITTEE[5]
LOAD-value >F,F) 'ANDMARY, William' 3594 (T,F,T,T,T,F)  'BACK, Helen' 8204
Next LOAD = tuple 4 COMMITTEE[1]
, >,(F,T,F,T,T,T) 'MINH, Song Lee' 3737 (T,T,F,T,T,F) $     !  The LOAD is     !+
                                                            !  ended with $.   !
*INFORMATION* 5 TUPLE(S) WILL BE LOADED
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   Now let's look at what is stored in this relation, using PRINT.     !+
     PRINT NEW.RELATION $A *     !   The $A asks for all values in a tuple,   !+
                                 !   and the asterisk (which in this posi-    !+
                                 !   tion means "null conditions") asks for   !+
                                 !   all tuples in the relation.              !
*INFORMATION* 5 TUPLE(S) WILL BE PRINTED
SNORKLEWHACKER, John       5335          T          F          F          F
     F          F
GRINSCH, Zephus       2620          F          F          T          T
F          F
ANDMARY, William       3594          T          F          T          T
 T          F
BACK, Helen       8204          F          T          F          T          T
       T
MINH, Song Lee       3737          T          T          F          T          T
          F
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   That wasn't very pretty.  The extension numbers could be aligned,   !+
     !   and the committee flags don't need that much space.  We can avoid   !+
     !   these problems with some options on the PRINT command.              !+
     PRINT/VFORMAT='1X,A,T26,I4,5X,6L3' $S $S $S     !   The $S's mean, "the  !+
                                                     !   same as last time"   !+
                                                     !   for the ra-spec,     !+
                                                     !   print-names and      !+
                                                     !   then conditions.     !
```

```
*INFORMATION* 5 TUPLE(S) WILL BE PRINTED
SNORKLEWHACKER, John     5335        T  F  F  F  F  F
GRINSCH, Zephus          2620        F  F  T  T  F  F
ANDMARY, William         3594        T  F  T  T  T  F
BACK, Helen              8204        F  T  F  T  T  T
MINH, Song Lee           3737        T  T  F  T  T  F
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   That looks better.  If we want to add to this list, we can LOAD it  !+
     !   again.  New tuples will be put at the end unless we use an option   !+
     !   to say where we need to begin.  Tuples can be deleted with the      !+
     !   UNLOAD command.  If you want to save the new version you must PUT   !+
     !   it back into the database.  Otherwise it is lost when you EXIT      !+
     !. from ACE.                                                            !+
     LIST/FULL *.*;*     !   This will show us everything in the workspace.   !+
                         !   For some files it will also give additional      !+
                         !   information which we recorded previously.        !
SPEC                                          CLASS      ENTRIES   NUNITS   CUNITS
DATASPEC              TEXTSPEC              PO    PN
KEYWORDS
TITLE
MESH:1.A;1                                     RELATION        4       177       200
IAC000012.IAC;1                               RWDL  RWDL
TEST
TEST DATA FOR SUPERTAB MACROS
MESH:1.B;1                                     RELATION        4       813       200
IAC000013.IAC;1                               RWDL  RWDL
TEST
TEST DATA FOR SUPERTAB MACROS
TEST:1.REL;3                                   RELATION       36       507      1650
IAC000020.IAC;1                               RWDL  R
TEST,VALUES
Test values of all types, lengths and dimensions (incl. variable)
NEW:1.RELATION;1                               RELATION        5         0       200
*                                             *     *


NEW:1.RELATION;2                               RELATION        5        40       276
*                                             *     *


*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   The blank lines and asterisks above indicated a lack of, or the     !+
     !   defaults for, the corresponding items.  NEW.RELATION is missing     !+
     !   owner/nonowner protections, database file spec (we haven't PUT it   !+
     !   out yet) and descriptive words and title.  We don't really want it  !+
     !   so let's delete it.  The DELETE command requires the version num-   !+
     !   bers (the semicolon and following number) or wild card for them.    !+
     DELETE NEW.RELATION;*
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   Occasionally you need to know what's going on in the "outside"      !+
     !   world:  check on the status of a batch job, send notes to someone   !+
     !   or find out why the computer is so slow.  ACE lets you do this      !+
     !   without exiting and losing your workspace, with the HOST command.   !+
     HOST '$SHOW QUE SYS$PRINT'

  * Generic Device queue "SYS$PRINT"  Burst Flag
```

```
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   Some commands use "conditions" to select a bunch of tuples from a  !+
     !   datastructure.  These include PRINT, UNLOAD and TALLY.  Conditions  !+
     !   can be very simple or quite complex.  One thing you should know     !+
     !   that will help a lot is that each tuple has an index-number asso-   !+
     !   ciated with it called $I (or for the ARRAY class, $I0, $I1, etc.    !+
     !   for each index of the array).  Conditions can include reference to  !+
     !   the tuple numbers with this.  Some examples, using TALLY because    !+
     !   it doesn't give too much printout, follow.                          !+
     TALLY MESH.A $I LE 15 AND NODE GT 3
 14 TUPLE(S) SATISFY THE CONDITIONS.
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >TALLY $S X IN (-4 .. 0.02, 1, 1.03..1.66)
 5 TUPLE(S) SATISFY THE CONDITIONS.
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >TALLY $S  X LEN Y AND Y NE $MAX     !   Note the use of LEN instead of   !+
                                         !   LE when comparing two names.     !
 39 TUPLE(S) SATISFY THE CONDITIONS.
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   ACE can sort values in a datastructure of the class RELATION.  You  !+
     !   have a number of choices with the SORT command to give you the      !+
     !   result you want.  SORT will order nonscalar items in a reasonable   !+
     !   manner also, even if they have variable dimensions, by using an     !+
     !   analogy to alphabetic order:  All first elements are compared, and  !+
     !   for those that are equal, the next elements are compared, and so    !+
     !   on.  If all elements are equal when one of the two items runs out   !+
     !   and the other has more, the other is judged to be "greater."  For   !+
     !   our example TEST.REL will be used.  Its structure is:               !+
     SHOW FILE TEST.REL
DATASTRUCTURE=TEST:1.REL;3, CLASS=RELATION
TUPLES     NAME       DIMENSIONS TYPE  FORMAT
       4 A                        I1    1X,I4
         AA           3          I1    1X,I4
         AAA          *          I1    1X,I4
         B                       R1    F8.2
         BB           3          R1    F8.2
         BBB          *          R1    F8.2
         C                       R2    D11.2
         CC           3          R2    D11.2
         CCC          *          R2    D11.2
         D                       Z2    2H [,2F8.2,1H]
         DD           2,2        Z2    2H [,2F8.2,1H]
         DDD          2,*        Z2    2H [,2F8.2,1H]
         E                       Z4    2H [,2D11.2,1H]
         EE           2,2        Z4    2H [,2D11.2,1H]
         EEE          2,*        Z4    2H [,2D11.2,1H]
         F                       L1    1X,L2
         FF           3          L1    1X,L2
         FFF          *          L1    1X,L2
         G                       C6    1X,A6
         GG           3          C6    1X,A6
         GGG          *          C6    1X,A6
         H                       C*    1X,A
         HH           3          C*    1X,A
         HHH          *          C*    1X,A
```

```
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >SORT TEST.REL B
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >PRINT TEST.REL B *      !   SORT does not create a new version of the      !+
                             !   file it sorts.  Otherwise large files could    !+
                             !   not be sorted in the workspace.                !
*INFORMATION* 4 TUPLE(S) WILL BE PRINTED
  -1.10
   0.40
   1.01
  10.10
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >SORT /ZRI $S DD[..],B/HILO      !   B will be sorted high to low as a      !+
                                     !   secondary sort to the sorting of the   !+
                                     !   complex vector DD.  The /ZRI at the    !+
                                     !   beginning makes REAL, then IMAGINARY   !+
                                     !   the default for complex number sorts.  !
*INFORMATION* LAST ACE COMMAND PROCESSED

ACE >!   Modules can be executed with the RUN command.  A special one exists !+
     !   for executing executable programs that are available as host files  !+
     !   on the current user directory.  Its format is similar to all of     !+
     !   the analysis program modules.  If the program is written in FORTRAN !+
     !   it can access parameters, IAC utilities and ACE common blocks.      !+
     RUN USER (EXE=EXAMPLE)
BEGIN USER        RUNID=13420339

    This is a FORTRAN program
    that has been compiled and
    linked into an executable
    module.
FORTRAN STOP
END USER
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   The SUBMIT command is similar to RUN, but puts the job into one of  !+
     !   the batch queues on the computer.  Options can control time, which  !+
     !   queue to enter, and other attributes you could specify for the job. !+
     SUBMIT AFTER=17:30 'RUN USER (EXE=EXAMPLE)'
  Job 9289 entered on queue SYS$BATCH
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >!   You may find it difficult to remember everything about ACE.  One     !+
     !   "trick" you can use when you aren't sure what might come next in    !+
     !   the command is to make a deliberate mistake at that point.  You     !+
     !   see the *ERROR* The token at the end of:... message, and it lists   !+
     !   what it would have permitted:                                       !+
     OPEN (USER=JG, $XXXXX      !   The nice thing about $XXXX is that it     !+
                                !   is guaranteed NOT to match at arbitrary   !+
                                !   points in the command.  It IS an error.   !
*ERROR* The token at the end of:
        OPEN ( USER = JG , $XXXXX
        did not satisfy any of the following prompts:
 DIRECTORY     STATUS     ACCESS
DIRECTORY=directory >D='[JG]',A=NOCONC)     !   And off we go.                  !
*INFORMATION* LAST ACE COMMAND PROCESSED
ACE >E     !   E is a nice abbreviation for EXIT.  The closing up of the      !+
           !   ACE program takes a while as the computer digests the JCL...   !
*INFORMATION* LAST ACE COMMAND PROCESSED
*INFORMATION* CONDITIONS DETECTED DURING THE RUN
                  WARNING  ERROR  ABORT
                        0      1      0
%NONAME-E-NOMSG, Message number 00000002
END ACE
```

A.5 EXPLANATION OF ACE CONTROL COMMANDS

```
ACE >%Z
*INFORMATION* The control commands include:
                A - move to an alternate node prompt
                B - back to repeated or sibling node
               *C - change input token(s) associated with this node
                D - down to child node
                E - show examples for this node
                F - forward to next repeated or sibling node
                H - show help for this node
                I - go to INPUT mode
                K - kill the command
                P - move to previous data token node
               *R - reject this node
               *S - show node properties
                U - up to parent node
                V - view token(s) supporting this node
                X - go to EXPLORATORY mode
                Z - provide help on control commands

              * not fully operational
```

A - When you don't want to answer a prompt, this control will find another prompt that you might want. If you exhaust the list of possibilities, it will recycle.

B - Go back to the prompt for the node in the syntax tree that is to the left of the current node and at the same level. If there is not such a node to the left, remain at the current node. When dealing with a tree built while examining command tokens, you may end up with the same node prompt but for a previous token (if the node has (*) on the chart).

D - Go down to the prompt for the first node below this one in the syntax tree. If this one is at the bottom, stay there.

E - Show a few examples for acceptable replies to this prompt.

F - Go forward to the prompt for the node in the syntax tree that is to the right of the current node and at the same level. (See "B" for similar effects.)

H - Provide help in answering the prompt. Sometimes by going U or D you can get further explanation with Help there.

I - Go to INPUT mode (from Exploratory mode). The next prompt you see may be different from this one. You may then enter appropriate command tokens.

K - Kill the command. Use this when you are thoroughly confused and can't seem to correct a problem in entering your command.

P - Go to the node prompt for the TOKEN entered prior to this node. This will seem to skip around in the tree a bit in reference to the command chart. Its basis is the command you have entered so far.

U - Go up the syntax tree to the "parent" node prompt. If at the top (the prompt is "ACE >"), stay there.

V - View those tokens that have been entered for prompts at or below this node. This reviews a part of your command.

X - Switch to EXPLORATORY mode (from Input mode). All autoinput until you switch back will be "%" so that your commands would all be control commands. In this mode you can move through the syntax tree, both what you already have created (or for what you MIGHT have done) and what you have left as possibilities. You cannot modify Input-mode command tokens. When you return to Input mode, you will be back at the Input node prompt at which you left.

Z - Display a brief review of the control commands (one line each).

A.6 HELP AND EXAMPLES FOR ACE

```
ARCHIVE-statement >ARCHIVE %HE
ARCHIVE archives database cataloged files, by resetting STATUS to A:
  ARCHIVE ARCHIVE-qualifiers file-specs
The user has responsibility for physically removing the file from the database
directory and disposing of the file, after archiving.
ARCHIVE
ARCHIVE COMP.DATA;3
ARC/V XY.*;*
```

```
CATALOG-statement >CATALOG %HE
CATALOG enters an IAC file description into the database catalog:
  CATALOG CATALOG-qualifiers new-file-spec file-class
  list(user-catalog-attributes)
The data and/or text host files must exist in the database directory before
cataloging.  The user-defined catalog attribute defaults are
DATASPEC=' ', TEXTSPEC=' ', PO='*', PN='*', KEYWORDS=' ', TITLE=' '.
CATALOG
CATALOG OTHER COMP.DATA
CATA/V XY.Z;2 REL (DATA=DXY.DAT, PN=R, PO='*')
```

```
CHANGE-statement >CHANGE %HE
CHANGE alters tuple values in a relation or array:
  CHANGE  ra-spec  changes  conditions
A change equation may mix only data types Cn and C*, not others (not I1 and R1,
R1 and R2, etc.).  If the left hand side of a change equation is a scalar,
the right hand side must be scalar.  If the left hand side is a nonscalar,
then the right hand side must consist of one or more vectors.
    The following change equation restrictions apply for an attribute having
an "*" definition, i.e. a variable index or substring size.
  1)  The attribute may be referenced by only one left hand side.
  2)  The attribute may not be referenced by a right hand side, if it has
      been referenced by a previous left hand side.
CHANGE
CHANGE  ACCOUNT.REC  HOURS=60  NAME EQ BOB AND WEEK EQ 40
```

```
CLEAR-statement >CLEAR %HE
CLEAR clears user-definable parameters (sets values to standard).
The following alternates are identified by secondary keyword.
  CLEAR IO clears file specs for executive input and output:
    CLEAR IO
  CLEAR MACRO clears parameters for MACRO operation:
    CLEAR MACRO
  CLEAR MESSAGE clears message controls for information and warning messages:
    CLEAR MESSAGE
CLEAR
CLEAR MACRO
```

```
COST-statement >COST %HE
COST displays host accounting information - may be installation dependent.
COST


DEFINE-statement >DEFINE %HE
DEFINE creates a new file cataloged in the workspace.
The following alternates are identified by secondary keyword.
  DEFINE RELATION creates a relation data structure:
    DEFINE RELATION /RULES=conditions new-rel-spec rel-att-defs
  DEFINE ARRAY creates an array data structure:
    DEFINE ARRAY /RULES=conditions new-arr-spec list(arr-index-sizes)
    core-att-defs
The user-defined catalog attributes for the new file are initialized as
DATASPEC='*', TEXTSPEC=' ', PO='*', PN='*', KEYWORDS=' ', TITLE=' '.
DEFINE
DEFINE REL R.NEW R.OLD
DEFINE REL R.NEW A.OLD/2
DEF R A.TAB (AN1,R1/F10.3 AN2 R2/D12.4)
DEF ARRAY TRUSS.TEMP  X.NODES,FIRST_SET.TIME  TEMP,R1/F10.4


DELETE-statement >DELETE %HE
DELETE deletes cataloged files from the workspace or database:
  DELETE DELETE-qualifiers file-specs
DELETE
DEL B.TEMP
DEL/VERIFY/D *.DAT


EXIT-statement >EXIT %HE
EXIT terminates the ACE run:
  EXIT
EXIT


GET-statement >GET %HE
GET moves cataloged files from database.  Structured data files are
moved to workspace.  Text files and unstructured data files are moved
to user directory:
  GET GET-qualifiers file-specs
GET
GET NODES.X
GET/TEXT TRUSS.*


GETC-statement >GETC %HE
GETC copies the database catalog to the workspace, with user defined spec:
  GETC GETC-qualifiers new-rel-spec
GETC
GETC QUERY.CAT
GETC/V C.C;3
GETG-statement >GETG %HE
GETG processes data from a 'generic' character-formatted host file, and
inserts selected data items into a new relation in the workspace:
  GETG GETG-qualifiers host-file-ref /edit-file-ref new-rel-spec rel-att-defs
GETG
GETG X.DAT X.REL (NODE,I1 X,R1/F10.3 Y,R1/F10.3)
GETG/VERIFY GENERIC.DAT A.B X.Y
```

```
GETS-statement >GETS %HE
GETS copies a structured data file from a host directory to the workspace:
  GETS GETS-qualifiers host-file-spec new-file-spec
GETS
GETS X.DAT X.DAT
GETS/V X.DAT COMP.DATA


HOST-statement >HOST %HE
HOST executes a sequence of host operating system commands:
  HOST HOST-commands
HOST
HOST '$DELETE *.OBJ;*'
HOST('$PURGE', '$DIR')
HOST $S


JOIN-statement >JOIN %HE
JOIN horizontally joins two relations to form a third:
  JOIN rel-spec1 rel-spec2 JOIN-names new-rel-spec
JOIN
JOIN XY.DAT Z.DAT NODE XYZ.DAT
JOIN  OLD.THINGS NEW.THINGS  OLDATT/NEWATT  ALL.THINGS


LINK-statement >LINK %HE
LINK invokes the operating system's Linker:
  LINK LINK-parms
LINK
LINK(MOD=DISCOS,EXE=TEST,OBJ=CONTRL)
LINK(EXE=USERACE,QUAL=('MAP/FULL',CROSS),MOD=ACE+
      ,OBJ=(ACE,GRUNCX),MAIN=NEW)
LINK EXE=SUM.EXE,LIB='SUMLIB/LIB',OBJ=SUMOBJ.OBJ


LIST-statement >LIST %HE
LIST displays selected catalog attributes for workspace or database files:
  LIST LIST-qualifiers file-specs
LIST
LIST A.DAT
LIST *.*
LIST/D *.THERMAL


LOAD-statement >LOAD %HE
LOAD loads (adds) tuples into a relation:
  LOAD LOAD-qualifiers rel-spec /start-index-value LOAD-sequence
LOAD/NE X.DAT
LOAD NODES.DAT 5,1.0,0.,2.5 15,2.4,0.,6.2 $
LOAD HOURS.DAT /231 JONES,8 SMITH,32 $
LOAD /LA/NE NEW.DAT 1 (0.,3.) 2 (4.,2.5)


MERGE-statement >MERGE %HE
MERGE modifies an existing data structure by merging in another existing
data structure.  The following alternates are identified by secondary keywor
  MERGE RELATION vertically expands an existing relation:
    MERGE RELATION rel-spec /start-index-value MERGE-rel-spec
MERGE REL GLOBAL.NODES /1 LOCAL.NODES
```

```
OPEN-statement >OPEN %HE
OPEN opens an old or new database:
  OPEN list(OPEN-parms)
OPEN
OPEN (U=JONES,D='[XYZ.D2]')
OPEN(U=DAN,D='[DAN.DB]',S=NEW)


PLOT-statement >PLOT %HE
PLOT generates a plot file for display of graphs or charts:
  PLOT BEGIN PLOT-parms END
PLOT
PLOT BEGIN REL A.DAT NAMES(NODE,FORCE) HOSTFILE A.PLT END
PLOT BEGIN+
RELATION RUN3.STRESS+
HOSTFILE RUN3.STR+
NAMES (ELEM,X1,S2,S23)+
CURVES (1,X-AXIS=ELEM,Y-AXIS=S1) (2,X-AXIS=ELEM,Y-AXIS=S2) (3,X-AXIS=ELEM,Y-AXIS
=S23)+
LAYOUT (X-AXIS=CONSTANT,BACKGR=NONE)+
END


PRINT-statement >PRINT %HE
PRINT displays selected attributes and tuples from a relation or array:
  PRINT PRINT-qualifiers ra-spec PRINT-names conditions
PRINT
PRINT NODES.DAT X,Y,Z NODE IN (101..150)
PRINT TRUSS.TEMP NODE,TEMP  *


PROJECT-statement >PROJECT %HE
PROJECT projects parts of a relation or array file to a new file:
  PROJECT ra-spec PROJECT-names conditions new-ra-spec
PROJECT
PROJECT MESH.DAT $A (X IN 100. .. 450.) MESH2.DAT
PROJECT  ARRAY.C3  ($A0,$A1,X2,Y2) (Z2 LT 60.) ARRAY.C2


PUT-statement >PUT %HE
PUT moves cataloged files to database.  Structured data files are
moved from workspace.  Text files and unstructured data files are moved
from user directory:
  PUT PUT-qualifiers file-specs
PUT
PUT NODES.X
PUT/TEXT TRUSS.*


PUTS-statement >PUTS %HE
PUTS copies a structured data file from the workspace to a host directory:
  PUTS PUTS-qualifiers file-spec new-host-file-spec
PUTS
PUTS X.DAT X.DAT
PUTS/V COMP.DATA X.DAT
```

```
REDEFINE-statement >REDEFINE %HE
REDEFINE redefines a relation or array cataloged in the workspace:
  REDEFINE /RULES=conditions ra-spec list(att-redefs)
Any parts of the relation or array not redefined remain as before.  There are
two cases for redefinition:
1) The file has no tuples.  In this case any of the attribute definition
   characteristics (name, subscript, type, format) may be redefined.
2) The file has existing tuples.  In this case the name and format may be
   redefined.  Also the subscript may be redefined, provided that the number
  of elements per attribute is not changed.  This provision requires that
  a) the product of the fixed index sizes remains constant; and b) the number
  of variable indexes remains constant.  Note that correlation between old
  and new variable indexes is based on their corresponding orders.
  As examples, a subscript may be redefined from [6] to [3,2], from
  [*,6] to [3,*,2], from [1,*] to [*], or vice versa.
An attribute redefinition which contains "*" for the name, type or format,
respectively, indicates that that part of the current definition is not to be
changed.
REDEFINE
REDEFINE NODES.DAT (NODE=*,*/I5 X=*,R2/*)
REDEF P.Q A=B,C10/*


RELOAD-statement >RELOAD %HE
RELOAD reloads (modifies) existing values associated with index 0 of an array:
  RELOAD RELOAD-qualifiers arr-spec /RELOAD-names /factors conditions
  /prompt-names RELOAD-sequence
RELOAD NODES.DAT  NODE EQ 23  23,50.,120.,0. $
RELOAD $S/X  NODE IN(24..27) 50.,55.,60.,65. $


RENAME-statement >RENAME %HE
RENAME renames cataloged files in the workspace or database:
  RENAME RENAME-qualifiers file-specs RENAME-file-spec
RENAME
RENAME X.DAT;* Y.DAT;*
REN/D/V X*Y.DAT *.INFO


REORDER-statement >REORDER %HE
REORDER reorders a relation or array via index operations:
  REORDER ra-spec/arr-index-id REORDER-sequence
REORDER
REORDER RELATIONX.DAT  MOVE 51..100 BEFORE 400
REORDER ARRAYQ.DAT/1  SWAP 1,2,3 WITH 4,5,6 /*,6


RUN-statement >RUN %HE
RUN executes a module:
  RUN module-name-and-parms
RUN NASTRAN(F=PANEL,RFA=IACDMAP1,PRT=YES)
```

```
SET-statement >SET %HE
SET sets values for various parameters.  The following alternates are
identified by secondary keyword.
  SET IO sets file specs for executive input and output:
    SET IO IO-parms
  SET MACRO sets parameters for MACRO operation:
    SET MACRO MACRO-parms
  SET MESSAGE controls the display status of executive messages:
    SET MESSAGE list(message-controls)
  SET UCATT sets user catalog attributes for files in workspace or database:
    SET UCATT SET-UCATT-qualifiers file-specs list(user-catalog-attributes)
SET
SET MACRO
SET UCATT/D/V *.* (PO='*', PN=R)
SET MES INFO=ON

SET-IO-statement >SET IO %HE
SET IO sets file specs for executive input and output:
  SET IO IO-parms
IO parameters may be cleared (set to standard values) by the
"CLEAR IO" command, and displayed by the "SHOW IO" command.
SET
SET IO
SET IO (OUT=BULK.LIS, J=SAVE.DAT/OLD)
SET IO $S
SET IO (JOURNAL=' ', MESSAGE='*')

SET-MACRO-statement >SET MACRO %HE
SET MACRO sets parameters for MACRO operation:
  SET MACRO MACRO-parms
MACRO parameters may be cleared (set to standard values) by the
"CLEAR MACRO" command, and displayed by the "SHOW MACRO" command.
SET
SET MACRO
SET MAC (REL=XDAT.R, ID=X, T=DAT)
SET MAC $S

SET-MESSAGE-statement >SET MESSAGE %HE
SET MESSAGE controls display of executive messages of an informational or
warning nature:
  SET MESSAGE list(message-controls)
The status of the controls can be set ON or OFF for each message level.
The levels are:
  INFORMATION contains messages not relating to errors.
  WARNING concerns errors that may limit but not prevent command functions.
The settings can be displayed with the "SHOW MESSAGE" command or cleared by
the "CLEAR MESSAGE" command to get standard status at each level.
SET MESSAGE
SET MES INFO=OFF

SET-UCATT-statement >SET UCATT %HE
SET UCATT sets user catalog attributes for files in workspace or database:
  SET UCATT SET-UCATT-qualifiers file-specs list(user-catalog-attributes)
SET
SET UCATT
SET UC/D/V *.* (PO=*, PN=R)
```

```
SHOW-statement >SHOW %HE
SHOW displays characteristics or structure of an entity.
The following alternates are identified by secondary keyword.
  SHOW FILE displays structure of relations and arrays:
    SHOW FILE ra-specs
  SHOW IO displays file specs for executive input and output:
    SHOW IO
  SHOW MACRO displays current parameters for MACRO operation:
    SHOW MACRO
  SHOW MESSAGE displays current executive message controls by level:
    SHOW MESSAGE
SHOW FILE
SHOW MACRO
SHO FILE NODE.DAT,XYZ.*


SORT-statement >SORT %HE
SORT sorts a relation or array via comparison of attributes:
  SORT SORT-qualifiers ra-spec SORT-names
SORT
SORT X.DAT X
SORT/HILO STRESS.DAT (ELEM,X,Y)


SUBMIT-statement >SUBMIT %HE
SUBMIT initiates a batch ACE run:
  SUBMIT SUBMIT-parms list(input-strings)
SUBMIT
SUBMIT 'RUN NASTRAN (F=PANEL,RFA=IACDMAP1,PRT=YES)'
SUBMIT(QUE='SYS$HI',PRT=NO) 'BEGIN RUN INDA(FN=TRUSS2.F40+
      ,D=TRUSS2,MODE=(1,2,5)) COST   END'
SUBMIT 'RUN NASTRAN(F=TRUSS)', COST


TALLY-statement >TALLY %HE
TALLY displays count of selected tuples from a relation or array:
  TALLY ra-spec conditions
TALLY
TALLY NODES.DAT  X IN(100. .. 200.)


UNARCHIVE-statement >UNARCHIVE %HE
UNARCHIVE unarchives database cataloged files, by resetting STATUS to N:
  UNARCHIVE UNARCHIVE-qualifiers file-specs
The user has responsibility for physically providing the file in the database
directory, before unarchiving.
UNARCHIVE
UNARCHIVE COMP.DATA;3
UNARC/U XY.*;*
```

```
UNCATALOG-statement >UNCATALOG %HE
UNCATALOG removes an IAC file description from the database directory:
  UNCATALOG UNCATALOG-qualifiers file-specs
The data and/or text host files remain in the database directory after
uncataloging.
UNCATALOG
UNCAT COMP.DATA.3
UNCAT/V XY.*;*


UNIQUE-statement >UNIQUE %HE
UNIQUE identifies duplicate tuples in a relation or array:
  UNIQUE UNIQUE-qualifiers ra-spec UNIQUE-names
UNIQUE
UNIQUE MESH.DAT NODE
UNIQUE/PRINT/DEL POINTS.SPACE (X,Y)


UNLOAD-statement >UNLOAD %HE
UNLOAD unloads (deletes) tuples from a relation:
  UNLOAD UNLOAD-qualifiers rel-spec conditions
UNLOAD
UNLOAD/V CONFIGA.NODES  NODE IN(213..218)


WRITE-statement >WRITE %HE
WRITE writes specified format(s) and/or parameter(s) onto the OUTPUT file:
  WRITE WRITE-parms
Each WRITE format or parameter is written starting on a new record.
WRITE
WRITE '1H1,''THIS IS A STRING'',5X,''THIS IS TOO'''
WRITE(DATE,TIME)



simple-statement >%HE
A simple statement consists of a keyword, followed by other keywords, values
and/or symbols.  Alternate simple statements are identified by the initial
 keyword.  The following is a list of simple statement initial keywords.
  ARCHIVE     CATALOG     CHANGE      CLEAR       COST
  DEFINE      DELETE      EXIT        GET         GETC
  GETG        GETS        HOST        JOIN        LINK
  LIST        LOAD        MERGE       OPEN        PLOT
  PRINT       PROJECT     PUT         PUTS        REDEFINE
  RELOAD      RENAME      REORDER     RUN         SET
  SHOW        SORT        SUBMIT      TALLY       UNARCHIVE
  UNCATALOG   UNIQUE      UNLOAD      WRITE
%H
OPEN (U=SMITH
SHOW FILE A.DAT, TEST.MAT2
HOST
```

APPENDIX B - PLOT INFORMATION

This appendix defines the IAC plot file interface in terms of its format and card types.

B.1 PLOT FILE DEFINITION

Each card type permitted in the plot file is defined below, with its associated nametag and data fields. All card types except NAMES and VALUES are optional, depending upon the particular application. The nametag CONTINUE may be used an arbitrary number of times following any card type to continue that card. The nametag may start in any column, and may be abbreviated to unique leading characters. Other values are free field, with blank delimiters. Character values which contain blank, "=", ".." or "&&" must be enclosed within single apostrophes, and any enclosed apostrophe must be doubled. Keywords may be abbreviated to unique leading characters.

NAMES name1 name2 etc.

The NAMES card defines names for the variables (attributes) to be plotted. Each name field contains an alphanumeric name of maximum 10 characters, the first of which is alpha. (Alpha includes "A" through "Z", and underscore "_").

TYPES name1=type1 name2=type2 etc.

The TYPES card defines types for the variables to be plotted. Permissible types are I1 (integer), R1 (real single precision), R2 (real double precision), Z2 (complex single precision), Z4 (complex double precision), L1 (logical), Cn (character of length n), and C* (character of variable length). The variables may be given in any order, and any variable not given is assigned a default type R1.

FORMATS name1=format1 name2=format2 etc.

The FORMATS card defines formats for the variables. A format is any FORTRAN compatible format, exclusive of the outer parentheses, which generates a single record. The format is used to display values on the plot, e.g. along a constant axis (see LAYOUT card) or at a point (see CURVE card). The variables may be given in any order, and any variable not given is assigned a default format as follows.

Type	Format
I1	1X,I10
R1	1X,E10.3
Z2	1X,E10.3,1X,E10.3
Z4	1X,D10.3,1X,D10.3
L1	1X,L10
Cn	1X,C10
C*	1X,C10

VALUES value1 value2 etc.

Each VALUES card defines variable values which may be associated at a particular point on the plot. A value may be represented by any FORTRAN compatible constant, consistent with the variable type.

CURVE id parm1 parm2 etc.

Each CURVE card defines data and attribute parameters for a particular curve. Multiple curves may be defined to appear on a single plot. The id is any integer which identifies the curve. The parm fields may contain the following keyname=value form of parameters.

```
X-AXIS = name
Y-AXIS = name
POINT = name
SETP = set
LINE = linestyle          (default linestyle = SOLID)
MARKER = marker           (default marker = NONE)
VISIBILITY = visibility   (default visibility = Yes)
```

The X-AXIS, Y-AXIS and POINT parameters reference variables which are to be associated as labels with the X axis, Y axis, or point location, respectively. SETP operates in conjunction with the POINT specification, to allow only a subset of point locations to be labeled. LINE defines the type of connection between points, with linestyle being one of the values NONE, SOLID, DOT, DASH, DOT-DASH (or DASH-DOT). MARKER defines the point marker

(symbol) with marker being one of the values NONE, "+", "*", ".", "X" or "O". VISIBILITY specifies whether a curve and associated information will be visible, with visibility being one of the keyvalues YES or NO.>

The set value specifies a set of point numbers; it may be a single number of the form "i", a range of the form "i .. j", or an incremented range of the form "i .. j && k".

TITLE id title

Each TITLE card defines a text-string title which may appear on the plot. The id field defines the usage of the title on the plot, and is one of the following.

MAIN - main overall plot title (default is blank)
X-AXIS - X-axis title (default is blank)
Y-AXIS - Y-axis title (default is blank)
integer - alternate title

An arbitrary number of integer id's may be given; each defines an alternate title which may be interactively selected to replace the MAIN, X-AXIS or Y-AXIS title.

LAYOUT parm1 parm2 etc.

The LAYOUT card defines plot layout parameters. The parm fields may contain the following keyname=value form of parameters.

```
X-AXIS = axis-type                  (default type = LINEAR)
Y-AXIS = axis-type                  (default type = LINEAR)
BACKGROUND = background-type        (default type = STANDARD)
COORDINATES = coordinates-type      (default type = CARTESIAN)
```

The X-AXIS and Y-AXIS parameters define X and Y axis types, respectively. The corresponding type values may be CONSTANT (e.g. bar-type plot), LINEAR or LOGARITHMIC. BACKGROUND defines the resolution of the background grid, with the type being one of the values STANDARD, FINE, or NONE. COORDINATES defines the X-Y axis coordinates, with the type being one of the values CARTESIAN or POLAR (POLAR means X is radius, Y is counter-clockwise angle in degrees).

```
SCALE  parm1  parm2  etc.
```

The SCALE card defines scale parameters for the plot axes. The parm fields may contain the following keyname=value form of parameters.

```
XMIN = bound
XMAX = bound
YMIN = bound
YMAX = bound
XDIVISIONS = division-size
YDIVISIONS = division-size
SETX = set
SETY = set
```

The MIN, MAX, and DIVISIONS parameters define X and Y axis information (minimum value, maximum value and labeling value increment, respectively) for LINEAR or LOGARITHMIC scales. Defaults are such that "good" scales are generated. The SETX and SETY parameters define X and Y axis information for CONSTANT scales. The set value specifies a set of point numbers; it may be a single number of the form "i", a range of the form "i .. j", or an incremented range of the form "i .. j && k". The default set consists of all point numbers. The i and j values determine the axis lower and upper limits, respectively. For example, if i is less than 1, blank space will be created at the lower limit; the situation is similar if j is greater than the maximum point number. A k value larger than 1 may be used to skip axis labels on some of the intermediate points. If i is greater than j, and k is negative, points will be plotted in reverse order. If more than one variable is associated with a CONSTANT axis, a set of labels is generated for each.

```
LEGEND  id  legend
```

Each LEGEND card defines a text-string legend which may be interactively selected and positioned on the plot. The id is any integer which identifies the legend.

APPENDIX C - DISCOS ENHANCEMENTS

This appendix documents several IAC-associated enhancements which have been added to the DISCOS module.

Section C.1 defines the new nametag/id input-file layout. The nametag labels and associated free-field formatting improve the user's visibility and access to the data, and allow the input processor to locate most of the data errors during the initial execution. In addition, the various entities in the model (hinges, bodies, modes, nodes, sensors, momentum wheels) may be described using id's rather than sequential numbers; this facilitates more rapid changes to the configuration, without affecting the entire model.

Section C.2 describes generation of the A,B,C system definition matrices, and the associated procedure for linearization of the system equations. Labels are automatically generated within these matrices, and provide user visibility and access to the various hinge, body, mode, etc. type entities.

Finally, Section C.3 presents the interactive graphics capabilities which have been added to DISCOS. These provide for time-domain display of both topological and X-Y variable type plots. The displays may be generated either during the initial time-integration computational process, or later during a post-processing operation.

C.1 NAMETAG/ID INPUT-FILE LAYOUT

This section documents the new nametag/id input-file layout for DISCOS. The first topic describes the use of id's in defining and referencing the various entities of the model (hinges, bodies, modes, nodes, sensors and wheels). The following topics describe the input file rules, an example input file, and each of the nametag card types.

Id Definition and Usage - Id's are defined by their appearance in the input data. For example, a node id is defined by the value of the nid field on a NODE card in the bulk input file, or by a value of the NID attribute in the NODE type IAC relation file. An id is a positive integer value. The NDCO and IDCO functions, described below, allow a user to convert between id's and internal numbers.

The NDCO function routine has been provided to allow referencing of the various model entities via their id's. This routine is defined as a FORTRAN integer function:

```
INTEGER FUNCTION NDCO (STRING)
```

STRING is a given character string value which defines the id type and value, for example:

'H6' = hinge id 6

'B4' = body id 4

'M13B2' = mode id 13 on body id 2

'N725B10' = node id 725 on body id 10

'S3B4' = sensor id 3 on body id 4

'W2B5' = wheel id 2 on body id 5

The returned value of NDCO is the sequential (internal) number of the hinge, body, mode, etc.

By using the NDCO function within any user-defined DISCOS routines, the user can deal with more familiar quantities, and can be protected against modeling or reordering changes. For example, NASTRAN internal node numbers might be arbitrarily reordered to optimize its solution process, or DISCOS internal sensor numbers may all be changed if one body is added, deleted or modified. Once the internal number of an entity has been determined via NDCO, the standard DISCOS LENU and LOCU vectors, etc. may be employed as usual for necessary state vector definition and retrieval operations.

The IDCO function is identical in operation to NDCO, except that it performs the reverse operation. The given string contains numbers (e.g. S3B4 indicates sensor number 3 on body number 4), and the returned value of IDCO is the id of the hinge, body, mode, etc.

Input File Rules - The following rules apply for the DISCOS nametag/id input-file layout.

1) Each data line contains (a) a nametag starting in any column; and optionally (b) one or more integer, double precision and/or character values.

2) A nametag is an alphanumeric with first character alpha. Alpha includes "A" through "Z" and underscore "_" . The first 4 characters (or all characters, if less than 4) are required; additional characters may be given, but are ignored.

3) A value may be (a) an integer or double precision value, i.e. any value readable via a single FORTRAN I or F format; or (b) a character value, i.e. any other value. A character value is consistent with the FORTRAN free-field read convention. I.e., it may be enclosed within single apostrophes; if it contains blank, comma or apostrophe it must be so enclosed; and any enclosed apostrophe must be doubled.

4) A delimiter separates the nametag from the first value, and each value from any adjacent values. A delimiter consists of one or more commas and/or blanks. Examples of a valid delimiter are " "; ","; " "; " , "; and " ,, , ".

5) A comment line is any line which is blank, or whose first nonblank character is a "$". Comment lines may occur anywhere within the input file. They are ignored in data processing, except that they are included in the count of line numbers.

6) The nametag CONTINUE may be used an arbitrary number of times to continue the previous data line.

7) Data lines may appear in any order, except for lines associated with a block. A block begins with a line containing a header nametag, and ends with a line containing a trailer nametag. The trailer nametag is the header nametag catenated with END, e.g. BODY and ENDBODY. Other lines associated with the block may occur in any order between the header and trailer lines.

Input-File Example - An example of an actual input file is shown in Figure C.1-1.

```
RUN    TEST  'A. D. LIGHT'
TITLE   'TEST CASE FOR NAMETAG INPUT - TIME DOMAIN'
CONT    'SIMPLE FREE BEAM CASE DEVELOPED 3-82'
TEXT  NAMETAG.TXT
$
SOLUTION  NONLINEAR
ABORT  WARNING
DEBUG  YDOT CONTRL,EQADD,EXTOR
$
IPRINT  1
IPLOT  0
TSTART  0
TEND  .0031
TDELTA  .001
$
$ NDELTA, NCPARM, AND PARAMETER-DEFINITION PAIRS
CDATA  3,7   4,400.  5,.0446  6,1.  7,125.66
$
$ NOTE - PRIMARY SIZES SET VIA CARDS HINGE,BODY,NODE,MODE,SENSOR,WHEEL.
$
$ STARTING HINGE AND BODY (TO CONNECT INERTIAL TRIAD WITH BODY REF. PT.)
HSTART  1
BSTART  1
$
$ HINGE DEFINITIONS
HINGE  1,5,011010
HINGE  2,5,011111
$
BODY  1  LUMPED
  $ IACGET  NODE  BEAM.NODE
  NODE  1  0.,0.,0.
  NODE  2  1.,0.,0.
  NODE  3  2.,0.,0.
  NODE  4  3.,0.,0.
  NODE  5  4.,0.,0.
  NODE  6  5.,0.,0.
  NODE  7  6.,0.,0.
  NODE  8  7.,0.,0.
  NODE  9  8.,0.,0.
  NODE 10  9.,0.,0.
  NODE 11 10.,0.,0
  NODE 99  0.,1.,0.
  $ IACGET  NMASS  BEAM.NMASS
  NMASS  1  .02325  .01,0.,0.,0.,0.,0.  0.,0.,0.
  NMASS  2  .0465
  NMASS  3  .0465
  NMASS  4  .0465
  NMASS  5  .0465
  NMASS  6  .0465
  NMASS  7  .0465
  NMASS  8  .0465
  NMASS  9  .0465
  NMASS 10  .0465
  NMASS 11  .02325
  NMASS 99  0.
```

Figure C.1-1: DISCOS Example Input File

```
  $ IACGET   MODE   BEAM.MODE
  MODE  4   1    0 0   2.8362   0   1.3440  0
  CONT      2    0 0   1.5033   0   1.3105  0
  CONT      3    0 0   0.2491   0   1.1749  0
  CONT      4    0 0  -0.7987   0   0.8958  0
  CONT      5    0 0  -1.4990   0   0.4861  0
  CONT      6    0 0  -1.7455   0   0.0     0
  CONT      7    0 0  -1.4990   0  -0.4861  0
  CONT      8    0 0  -0.7987   0  -0.8958  0
  CONT      9    0 0   0.2491   0  -1.1749  0
  CONT     10    0 0   1.5033   0  -1.3105  0
  CONT     11    0 0   2.8362   0  -1.3440  0
  CONT     99    0 0   0.0      0   0.0
  $
  MODE  5   1    0 0   2.7023   0   2.2800  0
  CONT      2    0 0   0.4997   0   2.0477  0
  CONT      3    0 0  -1.2096   0   1.2649  0
  CONT      4    0 0  -1.8872   0   0.0536  0
  CONT      5    0 0  -1.3514   0  -1.0537  0
  CONT      6    0 0   0.0      0  -1.5003  0
  CONT      7    0 0   1.3514   0  -1.0537  0
  CONT      8    0 0   1.8872   0   0.0536  0
  CONT      9    0 0   1.2096   0   1.2649  0
  CONT     10    0 0  -0.4997   0   2.0477  0
  CONT     11    0 0  -2.7023   0   2.2800  0
  CONT     99    0 0   0.0      0   0.0     0
  $ IACGET   MTDISP   BEAM.THERMAL
  $ MTDISP   0.   4,.2   5,.1
  $ MTDISP   1.   4,.2   5,.1
  $ IACGET   MSTIF   BEAM.MSTIF
  MSTIF   4,4,1.2153E4   5,5,8.8701E4
  $ IACGET   MDAMP   BEAM.MDAMP
  MDAMP    4,4,1.0        5,5,1.0
  MIDISP   4,0.   5,0.
  MIVEL    4,0.   5,0.
  BHPOS   2,P   5   0.,0.,0.   11
  SENSOR   1    5   0.,0.,0.    3
  SENSOR   2    5   0.,0.,0.    7
  SENSOR   3    5   0.,0.,0.    1
ENDBODY
BODY   2   RIGID
  BMASS   .01   .001,.001,.001,0.,0.,0.   0.,0.,0.
  BHPOS   2,Q   5   0.,0.,0.   0.,0.,0.
ENDBODY
```

Figure C.1-1: DISCOS Example Input File (Continued)

Nametag Card Types - Each card type permitted in the input file is defined below, with its associated nametag and data fields. Note that the BODY and ENDBODY cards define a block, within which are grouped the data associated with a particular body.

RUN run-id user-name

The optional RUN card defines the run id (maximum 6 characters) and the user name (maximum 18 charactors). These values are printed in the heading at the beginning of each new page in the output-listing file. If the RUN card is not given, blank default values are used.

TITLE title1 title2

The optional TITLE card defines two title lines (maximum 72 characters each). These values are printed in the heading at the beginning of each new page in the output-listing file. If the TITLE card is not given, blank default values are used.

DEBUG routine1 routine2 etc.

The optional DEBUG card identifies routines for which DISCOS debug printout is desired during execution. Each routine value may be either (a) an integer value corresponding to a debug routine number; (b) an asterisk "*" indicating all debug routines; or (c) the name of a DISCOS standard routine for which debug has been provided.

TEXT file-spec

The optional TEXT card defines the spec of a character formatted text file, used for run documentation. Each line on this file is copied to the beginning of the output-listing file. An exception is a line which contains only the characters $PAGE in columns 1-5; this line is not copied, but causes the start of a new page in the output listing file.

ABORT condition

The optional ABORT card defines when the run is to be terminated. Only the first character of the string condition is meaningful: C causes termination after the input data has been checked, without executing the solution; W and E terminate the run in case warnings and errors, respectively, have been detected; and N attempts to avoid termination and to force execution even if errors have been detected. The default condition value is E.

SOLUTION solution-type

The SOLUTION card defines the problem solution type. Only the first character of the string solution-type is meaningful: N indicates nonlinear time domain, F indicates frequency domain, and L indicates linear time domain. (The user should be aware that linear time domain solutions within DISCOS are of questionable validity).

IPRINT interval

The optional IPRINT card defines the print interval for display of results. The integer interval gives the multiple of the integration time step. Default interval zero indicates no printing.

IPLOT interval

The optional IPLOT card defines the plot interval for display of results. The integer interval gives the multiple of the integration time step. Default interval zero indicates no plotting.

TSTART time

The optional TSTART card defines the starting time for a time domain solution. Default value for time is zero.

```
TEND  time
```

The TEND card defines the ending time for a time domain solution. The TEND card is ignored for a frequency domain solution.

```
TDELTA  increment
```

The TDELTA card defines the integration time step for a time domain solution. The TDELTA card is ignored for a frequency domain solution.

```
GRAVITY  gx  gy  gz  r
```

The optional GRAVITY card defines the effect of gravity and gravity gradient on the system. The values gx, gy, gz are projections of the gravity vector on the inertial X, Y, Z coordinate axes. The value r is the radius vector from the gravity source to the general vicinity of the body cluster.

```
CDATA  ndelta  ncparm  index,value  index,value  etc.
```

The CDATA card contains user-defined data associated with the DISCOS common /CONPAR/CNTDTA storage. The integer ndelta is the number of user-defined first order differential equations coded in subroutine CONTRL (number of control states). The integer ncparm is the total number of control parameters (ndelta + number of additional control parameters). First, ncparm locations at the beginning of the double precision CNTDTA vector are initialized to zero. Then for each index/value pair, the value is stored into CNTDTA at the location specified by the integer index.

```
UDATA  nudata  index,value  index,value  etc.
```

The optional UDATA card contains user-defined data associated with the DISCOS common /USEPAR/USEDTA storage. The USEDTA vector is similar to CNTDTA described on the CDATA card, but it provides a convenient separate storage area which is not affected by changes in the number of control variables, etc. First, nudata locations at the beginning of the double precision USEDTA vector are initialized to zero. Then for each index/value pair, the value is stored into USEDTA at the location specified by the integer index.

```
UFILES  spec1  spec2  etc.
```

The optional UFILES card defines user file spec(s) which may be used for transferring user data during DISCOS execution. A maximum of 5 specs may be given. The files are opened for reading and/or writing, and are assigned successive unit numbers 51 to 55.

```
HSTART  hid
```

The HSTART card defines the starting hinge. This is the hinge whose P-triad is the inertial coordinate triad and whose Q-triad is the reference triad of the starting body. (See HINGE and BSTART cards). The integer hid gives the starting hinge id.

```
BSTART  bid
```

The BSTART card defines the starting body. This is the body whose reference triad is the Q-triad for the starting hinge. (See HINGE and HSTART cards). The integer bid gives the starting body id.

```
HINGE  hid  etype  constraints
```

Each HINGE card defines a 6-degree-of-freedom hinge. The integer hid gives the hinge id. The integer etype gives the Euler rotation type, which defines the axis rotation sequence of the hinge Q-triad relative to the P-triad. This type may be either a 3-digit rotation sequence (e.g. 123), or a rotation number 1-12; the correlation between numbers and sequences is given by 1=123, 2=121, 3=131, 4=132, 5=231, 6=232, 7=212, 8=213, 9=312, 10=313, 11=323, 12=321. The constraints is a packed 6-digit integer, whose digits specify the constraint code for each of the respective axis 1,2,3 rotations and x,y,z translations. The code 0 indicates no constraint (free relative motion), 1 indicates fixed (no relative motion), and 2 indicates rheonomic (user-defined motion via subroutines ADT and ADDT).

```
HIDISP  hid,dof,disp  hid,dof,disp  etc.
```

Each HIDISP card defines one or more hinge initial displacements (Q-triad relative to P-triad). The integer hid gives the hinge id. The integer dof gives the freedom number (1-6, corresponding to the 3 rotations and 3

translations). The value disp is the initial displacement. Any hinge freedom not specified is assigned a default displacement zero.

HIVEL hid,dof,vel hid,dof,vel etc.

Each HIVEL card defines one or more hinge initial velocities (Q-triad relative to P-triad). The integer hid gives the hinge id. The integer dof gives the freedom number (1-6, corresponding to the 3 rotations and 3 translations). The value vel is the initial velocity. Any hinge freedom not specified is assigned default velocity zero.

```
BODY  bid  RIGID
```

Each of these BODY cards defines the start of a rigid body data group. (The ENDBODY card defines the end of the data group). The integer bid gives the body id. Only the first character R of the string RIGID is required.

```
BHPOS  hid  pqcode  etype  r1  r2  r3  x  y  z
```

Each of these BHPOS cards gives a body-to-hinge positioning definition for the rigid body. One BHPOS card is used for each hinge triad (P-triad or Q-triad) on the body, except for the starting hinge whose position is defined automatically by DISCOS. (See HSTART and BSTART cards). The integer hid gives the hinge id. Only the first character of the string pqcode is meaningful: P indicates P-triad and Q indicates Q-triad. The integer etype gives the Euler rotation type, which defines the axis rotation sequence used to position the hinge triad relative to the body reference triad. (See HINGE card). The values r1, r2, r3 are the positioning rotations about the respective 1st, 2nd, 3rd axes in the sequence. The values x, y, z are the body reference coordinates of the origin of the hinge triad.

```
SENSOR  sid  etype  r1  r2  r3  x  y  z
```

Each of these SENSOR cards defines a sensor for the rigid body. The integer sid gives the sensor id (unique within the body). The integer etype gives the Euler rotation type, which defines the axis rotation sequence used to position the sensor triad relative to the body reference triad. (See HINGE card). The values r1, r2, r3 are the positioning rotations about the respective 1st, 2nd, 3rd axes in the sequence. The values x, y, z are the body reference coordinates of the origin of the sensor triad.

```
WHEEL  wid  sid  axis  wtype  mass  rate
```

Each WHEEL card defines a momentum wheel for the rigid body. The integer wid gives the wheel id (unique within the body). The integer sid gives the id of a sensor, at which the wheel is located. Note that only one wheel may be located at a particular sensor. The integer axis gives the sensor triad axis (1, 2 or 3) about which the wheel spins. Only the first character of the string wtype is meaningful: C indicates constant speed wheel, and A or V indicates active (i.e. variable) speed wheel. The value mass is the wheel moment-of-inertia (must be larger than zero) about the spin axis. The value rate is the wheel initial spin velocity.

ENDBODY

The ENDBODY card defines the end of the body data group. (The BODY card defines the start of the data group).

```
BODY  bid  LUMPED
```

Each of these body cards defines the start of a lumped-mass flexible body data group. (The ENDBODY card defines the end of the data group). The integer bid gives the body id. Only the first character L of the string LUMPED is required.

```
NODE  nid  x  y  z
```

Each NODE card defines a node for the lumped-mass body. (See also the IACGET card). The integer nid gives the node id (unique within the body). The values x, y, z are the node coordinates. If trailing coordinates are omitted, they are assigned default zeros.

```
NMASS  nid  mass  ixx  iyy  izz  ixy  ixz  iyz  sx  sy  sz
```

Each NMASS card defines mass data for a node of the lumped-mass body. (See also the IACGET card). The integer nid is the node id. The value mass is the lumped mass at the node. The values ixx, iyy, izz, ixy, ixz, iyz are components of the node inertia tensor. The values sx, sy, sz are the node static mass moments. If a trailing value is omitted, it is assigned a default zero. Any node not specified has all values assigned default zero. Note that the node inertia matrix is computed by DISCOS as

$$\begin{bmatrix} mass & 0 & 0 & 0 & sz & -sy \\ 0 & mass & 0 & -sz & 0 & sx \\ 0 & 0 & mass & sy & -sx & 0 \\ 0 & -sz & sy & ixx & -ixy & -ixz \\ sz & 0 & -sx & -ixy & iyy & -iyz \\ -sy & sx & 0 & -ixz & -iyz & izz \end{bmatrix}$$

```
BHPOS  hid  pqcode  etype  r1  r2  r3  nid
```

Each of these BHPOS cards gives a body-to-hinge positioning definition for the lumped-mass body. One BHPOS card is used for each hinge triad (P-triad or Q-triad) on the body, except for the starting hinge whose position is defined automatically by DISCOS. (See HSTART and BSTART cards). The integer hid gives the hinge id. Only the first character of the string pqcode is meaningful: P indicates P-triad and Q indicates Q-triad. The integer etype gives the Euler rotation type, which defines the axis rotation sequence used to position the hinge triad relative to the body reference triad. (See HINGE

card). The values r1, r2, r3 are the positioning rotations about the respective 1st, 2nd, 3rd axes in the sequence. The integer nid gives the id of a node, at which the origin of the hinge triad is located.

```
SENSOR  sid  etype  r1  r2  r3  nid
```

Each of these SENSOR cards defines a sensor for the lumped-mass body. The integer sid gives the sensor id (unique within the body). The integer etype gives the Euler rotation type, which defines the axis rotation sequence used to position the sensor triad relative to the body reference triad. (See HINGE card). The values r1, r2, r3 are the positioning rotations about the respective 1st, 2nd, 3rd axes in the sequence. The integer nid gives the id of a node, at which the origin of the sensor triad is located.

```
WHEEL  wid  axis  wtype  mass  rate
```

Each WHEEL card defines a momentum wheel for the lumped-mass body. The field definitions on the card are the same as for the rigid body.

```
MODE  mid  nid,tx,ty,tz,rx,ry,rz  nid,tx,ty,tz,rx,ry,rz  etc.
```

Each MODE card defines a mode shape for the lumped-mass body. (See also the IACGET card). The integer mid gives the mode id (unique within the body). The integer nid gives the node id. The values tx, ty, tz give the 3 translational displacements at the node; and rx, ry, rz give the 3 rotational displacements. Note that DISCOS assumes all nodal displacements are in terms of the body reference triad. Any node not specified within a mode shape is assigned default displacements zero.

```
MSTIF  rmid,cmid,stiff  rmid,cmid,stiff  etc.
```

The MSTIF card defines the modal stiffness matrix for the lumped-mass body. (See also the IACGET card). The integers rmid and cmid give the row and column mode id, respectively. The value stiff is the modal stiffness. Any stiffness not specified is assigned a default zero.

```
MDAMP  rmid,cmid,damp  rmid,cmid,damp  etc.
```

The MDAMP card defines the modal damping matrix for the lumped-mass body. (See also the IACGET card). The integers rmid and cmid give the row and

column mode id, respectively. The value damp is the modal damping. Any damping not specified is assigned a default zero.

MIDISP mid,disp mid,disp etc.

The MIDISP card defines modal initial displacements for the lumped-mass body. The integer mid gives the mode id. The value disp is the initial displacement. Any initial displacement not specified is assigned a default zero.

MIVEL mid,vel mid,vel etc.

The MIVEL card defines modal initial velocities for the lumped-mass body. The integer mid gives the mode id. The value vel is the initial velocity. Any initial velocity not specified is assigned a default zero.

MTDISP time mid,disp mid,disp etc.

Each MTDISP card defines the modal thermal displacements at a particular time for a lumped-mass body. (See also the IACGET card). Additional information on modal thermal displacements is given in Section 4.4 and Appendix F. The value time gives the time at which the displacements are measured. The integer mid gives the mode id. The value disp is the thermal displacement. Any displacement not specified is assigned a default zero. Note that DISCOS computes the thermal displacement at any time via linear interpolation from the input values. No extrapolation is performed; if the time is outside the input time range, the closest input time value is used. The elastic (load producing) modal displacement is the total minus the thermal displacement.

IACGET tag-name IAC-file-spec

Each IACGET card within the lumped-mass body data group causes data to be retrieved from an IAC array or relation file. The string tag-name is one of the values NODE, NMASS, MODE, MSTIF, MDAMP, or MTDISP; it specifies that data associated with that nametag is to be obtained from the file rather than from in-stream data cards.

ENDBODY

The ENDBODY card defines the end of the body data group. (The BODY card defines the start of the data group).

CYCLE

Each of these CYCLE cards defines the start of a transfer function data group for a frequency domain solution. (The ENDCYCLE card defines the end of the data group).

LRY itype itfin jtfout kplot iaflg nb ib1 ib2 ib3

The LRY card defines frequency solution control data.

The integer itype gives the transfer function type:

1 = plant only (G)
2 = controller (H)
3 = open loop (GH)
4 = open loop (HG)
5 = closed loop (GH/(1+GH))
6 = closed loop
7 = psuedo open loop
8 = special case, allowing opening of single return loop

Note that a minus sign on itype=3, 4, 5 or 7 indicates negative controller feedback; a minus sign on itype=8 indicates completely closed loop transfer function.

The integer itfin is usually a local identifier for the transfer function input variable. It references (depending upon itype) either a sensor signal number or a controller output variable which is the V(IN) of the expression V(OUT)/V(IN)=TF. However, if itype=±8, itfin is a global reference output state of Y* (whose feedback loop is cut).

The integer jtfout is usually a local identifier for the transfer function output variable. It references (depending upon itype) either a sensor signal number or a controller output variable which is the V(OUT) of the expression V(OUT)/V(IN)=TF. However, if itype=±8, jtfout is a global reference input state of Y* (whose feedback loop is cut).

The integer kplot is a plot activation code:

0 = no plots
1 = 1 plot will be made

The integer iaflg specifies whether characteristic roots from the transfer function are to be selected from the characteristic matrix AR (default iaflg=0) or AR transposed (iaflg=1).

The integer nb gives the number of B variables to feed back, for itype=7. Maximum nb=3. The integers ib1, ib2 and ib3 give the respective local id's for variables fed back. Trailing unused values may be omitted.

```
IRY  rtexp  gtexp  rtexps  scon
```

The optional IRY card defines frequency solution control data. The integers rtexp, gtexp are the respective root, gain tolerance exponents. The integer rtexps is the root tolerance exponent used to remove shift frequency. The integer scon is the shift constant (CON) for subroutne NUMS; CON is set to minus square root of scon. A missing or zero value results in a default (-7, -7, -7, 3 for the respective values).

```
TITLE  plot-title
```

The optional title card defines a title line for all frequency solution plots. If the TITLE card is not given, a blank default value is used.

```
POLY
```

The POLY card requests that the transfer function be computed and printed as a ratio of polynomials.

```
EIGEN
```

The EIGEN card requests computation of all eigenvectors.

```
FPLOT  plot-type  fmin  fmax  dbmin  dbmax  amin  amax
```

Each optional FPLOT card requests a frequency plot. The plot-type keyvalue string specifies the display type:

```
        *     = no display, system characteristic roots are found (default)
        BODE  = Bode
        NICH  = Nichols
        NYQU  = Nyquist
        NINY  = Nichols and Nyquist
        BONN  = Bode, Nichols and Nyquist
```

Other values are as follows (omitted trailing values result in default zero).

```
        fmin  = frequency sweep lower limit
        fmax  = frequency sweep upper limit
        dbmin = minimum DB amplitude for Bode, Nichols plots
        dbmax = maximum DB amplitude for Bode, Nichols plots
        amin  = minimum amplitude for Nyquist plots
        amax  = maximum amplitude for Nyquist plots
```

RPLOT sra,sia,nrl,xmin,xmax,ymax,aloc,drv,dtv,thv sra,sia,etc. etc.

Each optional RPLOT card requests a root locus plot for one or more loci. For each locus, the following values are given.

sra = real part of starting area locator
sia = imaginary part of starting area locator
nrl = generate loci for the nrl poles nearest to the starting area locator
xmin = minimum real value to plot
xmax = maximum real value to plot
ymax = maximum imaginary value to plot (minimum = -ymax)
aloc = phase
drv = parameter (may be zero)
dtv = parameter (may be zero)
thv = parameter (may be zero)

ENDCYCLE

The ENDCYCLE card defines the end of the transfer function data group. (The CYCLE card defines the start of the data group).

C.2 A,B,C MATRICES

The DISCOS RUN parameter ABC=name may be used to generate and output the plant definition matrices associated with the linearized model. If the ABC parameter is specified, linearized model A, B and C matrices are generated and stored as IAC arrays in a host directory or IAC database (depending upon the value of the RUN parameter AREA); the arrays have the respective file specs name.A, name.B and name.C.

In order to utilize this capability, a DISCOS frequency response solution must be performed; the input data and user subroutines must establish the relationships between DISCOS state variables, control inputs and sensed outputs.

The A,B,C matrices are defined by

$$\dot{x} = Ax + Bu$$

$$y = Cx$$

where

x = DISCOS state vector, in the standard DISCOS defined order

u = user defined inputs (e.g. forces and torques)

y = user defined outputs (e.g. performance measures)

The sizes of the matrices are A(n,n), B(n,p) and C(q,n), where

n = number of states

p = number of inputs (user defined)

q = number of outputs (user defined)

The stored numerical data within the generated arrays is supplemented by alphanumeric labeling attributes, in order to improve the user's visibility and access to the data. The A, B, C arrays are shown in Figure C.2-1, and a summary of the labels is presented in Table C.2-1.

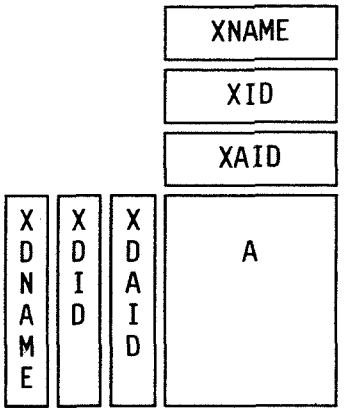
The user defined subroutines will generally include CONTRL, EQADD, EXTOR, KHINGE and SHAFTT.

In order to specify the vector of inputs u (e.g. the forces and torques impressed on the structure), the subroutines CONTRL, EXTOR, KHINGE and SHAFTT must establish the location and type of each input. The vector of p inputs is defined within DISCOS as the p control states (δ variables). The simplest method for defining these states is to define the p control equations

$$\dot{\delta}_i=\delta_i \qquad i=1,2,\ldots p$$

The control variables δ are impressed on the structure by passing them to one of the user routines (EXTOR for external forces and torques, KHINGE for gimbal torques, and SHAFTT for momentum wheel torques).

Sensed outputs y are defined in subroutine CONTRL as functions of plant states. These values are passed to EQADD and there loaded into the DISCOS YD vector in the standard manner. The order of the inputs u and the sensed outputs y is defined by the user via the order in which they are loaded into the YD vector in subroutine EQADD.

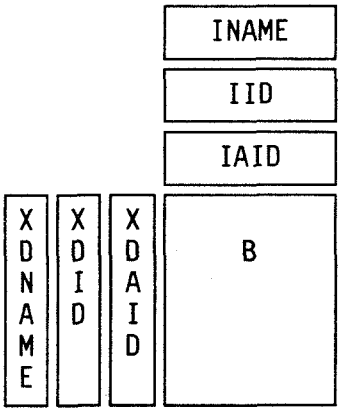


Description

A(N,N) = DISCOS A(1:N,1:N) and relates state derivatives to states.

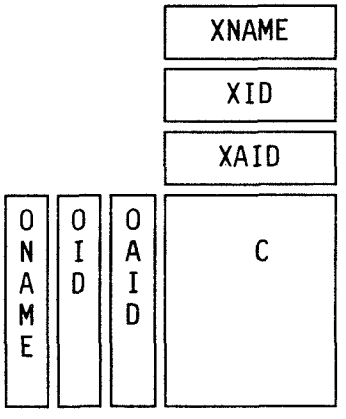
Attribute name/type

A	R2
XDNAME	C10
XDID	C10
XDAID	C10
XNAME	C10
XID	C10
XAID	C10



B(N,NDELTA) = DISCOS A(1:N,N+1:NX) and relates state derivatives to inputs.

B	R2
XDNAME	C10
XDID	C10
XDAID	C10
INAME	C10
IID	C10
IAID	C10



C(NXSS,N) = DISCOS AUX(1:NXSS,1:N) and relates outputs to states.

C	R2
ONAME	C10
OID	C10
OAID	C10
XNAME	C10
XID	C10
XAID	C10

Sizes: NX = number of independent coordinates in DISCOS state vector. NDELTA = total number of user defined equations in CONTRL. NXSS = total number of user defined "sensor signals" in CONTRL. NBTQ = total number of user defined "torque signals" in CONTRL. N = NX - NDELTA.

Note: "States" here means state perturbations, i.e. state values relative to reference states.

Figure C.2-1: DISCOS/IAC A,B,C Arrays

DISCOS Entity	NAME Attribute	ID Attribute	AID Attribute	Definitions
Body velocity states:				ID = entity identifier
ω_x	BRX	i	none	AID = associated identifier
ω_y	BRY	i	none	B = body
ω_z	BRZ	i	none	R = rotation
U	BTX	i	none	T = translation
V	BTY	i	none	X,Y,Z = axis
W	BTZ	i	none	i = body number (id)
Modal velocity states:				
ξ_m	MV	m	Bi	MV = modal velocity
ξ_m	MV	m	Bi	i = body number (id)
etc.	etc.	etc.	etc.	m = mode number (id)
Wheel velocity states:				
ω_w	WV	w	Bi	WV = wheel velocity
ω_w	WV	w	Bi	i = body number (id)
etc.	etc.	etc.	etc.	w = wheel number (id)
Mode displacement states:				
ξ_m	MD	m	Bi	MD = modal displacement
ξ_m	MD	m	Bi	i = body number (id)
etc.	etc.	etc.	etc.	m = mode number (id)

Note: For the nametag/id input-file option, id's instead of numbers are used for bodies, hinges, modes, etc.

Table C.2-1: A,B,C Matrix Labeling Attributes

DISCOS Entity	NAME Attribute	ID Attribute	AID Attribute	Definitions
Hinge displacement states:				H = hinge
β_h	HR1Y	h	PiQj	i = P body number (id)
β_h	HR2X	h	PiQj	j = Q body number (id)
β_h	HTX	h	PiQj	h = hinge number (id)
β_h	etc.	etc.	etc.	1,2,3 = rotation sequence
etc.				x,y,z = axis
Sensed outputs:				
	O	o	none	O = output
	O	o	none	o = output number
	etc.	etc.	none	
Inputs:				
	I	i	none	I = input
	I	i	none	i = input number
	etc.	etc.	none	

Table C.2-1: A,B,C Matrix Labeling Attributes (Continued)

C.3 GRAPHICS

The interactive graphics capabilites available within the IAC DISCOS module provide for plotting of topological and time-domain characteristics. Currently the DISCOS graphics capabilities are based on the Tektronix PLOT10 support package, and operate on the Tektronix 4000 series storage-tube display terminals.

DISCOS structural plots and X-Y curves can be interactively displayed either in the "realtime" (during DISCOS execution) mode or in the "batch" (DISCOS postprocessing) mode. The first two topics in this section describe the respective realtime and batch display operations. Plot parameters and specifications are supplied using the interactive menu selections described in the third topic. Input commands and results from a typical interactive graphics session are summarized in the final topic.

Structural plots consist of lines connecting each body reference frame with the rigid body position of all sensors and hinges associated with that body. Flexible displacements at each sensor and hinge are shown as dashed vectors. Viewing angles are supplied by the user to define the relationship of the DISCOS inertial axis frame with respect to the screen axis system. The screen axes are as follows.

axis 1 = out of screen, positive toward viewer
axis 2 = positive toward the right of the screen
axis 3 = positive toward the top of the screen.

Relationships between response variables can also be plotted in the form of X-Y curves. Table C.3-1 defines the list of response variables for which this plotting capability is currently available; shown are the variable name, the index range (each variable is treated as a vector), and the variable description.

More than one curve can be plotted on the same background grid. At present, only linear scales are available.

The plot capability is activated via the plot control parameter (IPLOT card for nametag input file layout, or IPDATA(2) for no-nametag layout). The IAC

command required to run DISCOS with interactive plotting depends upon whether a plot file is to be generated ("realtime" plots) or a previously created plot file is to be plotted ("batch" plots).

Realtime Plots - The term "realtime" refers to the creation of graphics output while DISCOS is executing. The ACE command to create realtime plots is of the form

```
RUN DISCOS (F = name, EXE = DISCOSX)
```

where "name" is the file containing the DISCOS input data (name.DAT) and DISCOSX is a user-generated version of DISCOS (DISCOSX.EXE) containing user-supplied subroutines. When the plot control parameter has been set and the ACE command to run has been given, DISCOS will process the input data and pause prior to integration. At this time a menu will appear on the screen by which the user can select the display mode and the type of plots (structural or X-Y plots) to be seen.

The default values of appropriate menu items are set to values generally required to create realtime plots. The execution of DISCOS can be temporarily interrupted to plot other response parameters calculated during that session using the "batch" mode or to continue the execution while observing a different set of parameters. During execution, DISCOS creates and saves a plot file (name.PTH) for use in subsequent "batch" plotting.

Batch Plots - The term "batch" refers to the creation (for interactive viewing) of graphical output from a plot file "name.PTH" previously created during a realtime job. The ACE command to create batch plots using a previously generated plot file is of the form

```
RUN DISCOS (F = name, EXE = DISCOSX, FP = spec)
```

where files "name" and "DISCOSX" are DISCOS input data and a user's version of DISCOS, respectively, and "spec" is the specification of the DISCOS plot file created by DISCOSX.EXE during a previous session.

The specification of a plot file in the RUN command causes some of the default menu parameters to be changed to those generally used for batch plots. Again, plot specifications are selected interactively using the graphics menus.

Graphics Menus - DISCOS graphics parameters and commands are selected interactively from menus which appear during execution. The graphics menus are shown in Figure C.3-1.

Menu items fall into two catagories: 1) current values of parameters, and 2) action options. To change the value of a parameter, select the menu item number corresponding to that parameter. Additional sub-menus will appear which will guide the user in establishing other parameters necessary for accomplishing the desired task. After all parameters for a given menu are established, the user then selects one of the action options which will either return to a lower level menu so that other parameters can be changed or will identify a course of action to the program.

If DISCOS is running in the realtime mode, i.e. calculating and plotting responses during the current session, the integration can be interrupted temporarily by typing "P" (for pause) on the keyboard (without a carriage return). DISCOS will complete the time step currectly in progress and then print "DISCOS IS IN A PAUSE STATE" on the screen. The user then has the option to continue the integration by typing "C" or to select the level zero menu by typing "M". Once the menu has been obtained, the user can plot other results up to that point (using the batch mode), alter some of the DISCOS input parameters, integrate to the next display or stop the run.

Typical Graphics Session - The interaction of the user with the graphics package is demonstrated through the listing of a typical graphics session in Figure C.3-2. This example plots results from the demonstration problem using a previously generated plot file, DISHDSCP.PTH, created by a version of DISCOS contained in file RGDISCOS.EXE and using DISCOS input data contained in file DISHDSCP.DAT. A structural plot is generated first followed by an X-Y plot.

Name	Index Range	Description
T,I	I=1	Time
Y,I	I=1,NEQ	State vector, Y
YD,I	I=1,NEQ	State vector time derivatives, YD
ALAM,I	I=1,NLAM	Interconnection constraint forces, LAMBDA
P,I	I=1, $\sum_{J=1}^{NB}$ (6+NMD(J))	Momentum corresponding to each D.O.F.
PMOM,I	I=1,6*NB	Contrib. of each body to tot. ang. & lin. momentum
HTOT,I	I=1,3	Total angular momentum vector
TOTL,I	I=1,3	Total linear momentum vector
ENGK,I	I=1,NB	Kinetic energy for each body
ENGP,I	I=1,NB	Potential energy for each body
AHTO,I	I=1	Total angular momentum
ATOT,I	I=1	Total linear momentum
TOTK,I	I=1	Total kinetic energy
TOTP,I	I=1	Total potential energy
TOTE,I	I=1	Total energy
AUX,I	I=1,NADD	Auxiliary variables
DDS,I	I=1,3*NS	X,Y,Z sensor displacement W.R.T. original position
Note:	NEQ =	number of state vector quantities
	NLAM =	number of constraint forces
	NB =	number of bodies
	NMD(J) =	number of modes for body J
	NS =	number of sensors

Table C.3-1: DISCOS X-Y Plot Variables

```
MENU LEVEL= 0.

 1  PICTURE DEFINITION =NONE
 2  PICTURE MODE =  REALTIME
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =CONTINUE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP


MENU LEVEL= 0. 1.

 1  PICTURE DEFINITION =NONE
 2  PICTURE DEFINITION =STRCTURE
 3  PICTURE DEFINITION =XYCURVE


MENU LEVEL= 0. 1. 2.

 1  BODIES =ALL
 2  TIME POINTS =   ALL
 3  VIEW AXES(R,S,T) =  +X      +Y      +Z
 4  VIEW ANGLES(RR,RS,RT) =  0.,0.,0.
 5  RIGID DISPLACEMET MAXIMUM = ACTUAL
 6  RIGID DISPLACEMET FACTORS(X,Y,Z) =   1.,1.,1.
 7  FLEXIBLE DISPLACEMENT MAXIMUM = ACTUAL
 8  FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =   1.,1.,1.
 9  SCALE = UNIFORM
10  INERTIAL COORDINATE AXES =  NO
11  BODY COORDINATE AXES =  NO
12  LABELS =NONE
13  MAXIMUM X,Y,Z = ACTUAL
14  MINIMUM X,Y,Z = ACTUAL
15  RETURN
```

Figure C.3-1: DISCOS Graphics Menu

```
MENU LEVEL= 0. 1. 2. 1.

 1  BODIES =ALL
 2  BODIES =USER



MENU LEVEL= 0. 1. 2. 2.

 1  TIME POINTS =   REALTIME
 2  TIME POINTS =   ALL
 3  TIME POINTS =   USER



MENU LEVEL= 0. 1. 2. 3.

 1  VIEW AXES(R,S,T) =  +X
 2  VIEW AXES(R,S,T) =  +Y
 3  VIEW AXES(R,S,T) =  +Z
 4  VIEW AXES(R,S,T) =  -X
 5  VIEW AXES(R,S,T) =  -Y
 6  VIEW AXES(R,S,T) =  -Z



MENU LEVEL= 0. 1. 2. 4.

VIEW ANGLES(RR,RS,RT) =



MENU LEVEL= 0. 1. 2. 5.

 1  RIGID DISPLACEMET MAXIMUM = ACTUAL
 2  RIGID DISPLACEMET MAXIMUM = USER
```

Figure C.3-1: DISCOS Graphics Menu (Continued)

```
MENU LEVEL= 0. 1. 2. 6.

RIGID DISPLACEMET FACTORS(X,Y,Z) =



MENU LEVEL= 0. 1. 2. 7.

 1  FLEXIBLE DISPLACEMENT MAXIMUM = ACTUAL
 2  FLEXIBLE DISPLACEMENT MAXIMUM = USER



MENU LEVEL= 0. 1. 2. 8.

FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =



MENU LEVEL= 0. 1. 2. 9.

 1  SCALE = UNIFORM
 2  SCALE = FULL



MENU LEVEL= 0. 1. 2.10.

 1  INERTIAL COORDINATE AXES =  NO
 2  INERTIAL COORDINATE AXES =  YES



MENU LEVEL= 0. 1. 2.11.

 1  BODY COORDINATE AXES =  NO
 2  BODY COORDINATE AXES =  YES
```

Figure C.3-1: DISCOS Graphics Menu (Continued)

```
MENU LEVEL= 0. 1. 2.12.

 1  LABELS =NONE
 2  LABELS =HINGES
 3  LABELS =SENSORS
 4  LABELS =BODIES
 5  LABELS =ALL


MENU LEVEL= 0. 1. 2.13.

 1  MAXIMUM X,Y,Z = ACTUAL
 2  MAXIMUM X,Y,Z = USER


MENU LEVEL= 0. 1. 2.14.

 1  MINIMUM X,Y,Z = ACTUAL
 2  MINIMUM X,Y,Z = USER


MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 T,1
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= NONE
 7  YTITLE= NONE
 8  DISPLAY TITLE = NONE
 9  DISPLAY SYMBOL= NONE
10  RETURN
```

Figure C.3-1: DISCOS Graphics Menu (Continued)

```
MENU LEVEL= 0. 1. 3. 1.

XYCURVE=



MENU LEVEL= 0. 1. 3. 2.

 1  XMAX  = ACTUAL
 2  XMAX  = USER



MENU LEVEL= 0. 1. 3. 3.

 1  XMIN  = ACTUAL
 2  XMIN  = USER



MENU LEVEL= 0. 1. 3. 4.

 1  YMAX  = ACTUAL
 2  YMAX  = USER



MENU LEVEL= 0. 1. 3. 5.

 1  YMIN  = ACTUAL
 2  YMIN  = USER



MENU LEVEL= 0. 1. 3. 6.

 1  XTITLE= NONE
 2  XTITLE= USER
```

Figure C.3-1: DISCOS Graphics Menu (Continued)

```
MENU LEVEL= 0. 1. 3. 7.

 1  YTITLE= NONE
 2  YTITLE= USER



MENU LEVEL= 0. 1. 3. 8.

 1  DISPLAY TITLE = NONE
 2  DISPLAY TITLE = USER



MENU LEVEL= 0. 1. 3. 9.

 1  DISPLAY SYMBOL= NONE
 2  DISPLAY SYMBOL= USER



MENU LEVEL= 0. 2.

 1  PICTURE MODE =  REALTIME
 2  PICTURE MODE =  BATCH



MENU LEVEL= 0. 3.

 1  SCREEN ERASURE =MANUAL
 2  SCREEN ERASURE =AUTO
```

Figure C.3-1: DISCOS Graphics Menu (Continued)

```
MENU LEVEL= 0. 4.

 1  ACTION  AFTER EACH PICTURE =CONTINUE
 2  ACTION  AFTER EACH PICTURE =PAUSE



MENU LEVEL= 0. 6.

 1  DELTAT =  0.20000D-02
 2  ENDT   =  0.50000D+01
 3  NOPRNT =  50
 4  NOPLOT = -50
 5  RETURN
```

Figure C.3-1: DISCOS Graphics Menu (Continued)

```
$ IAC
BEGIN ACE RUNID=13583596
Enter command, or EXIT to terminate, or %H for help.
ACE >OPEN(U=GATES,D='[TEST.SYS]')
ACE >RUN DISCOS(EXE=RGDISCOS,F=DISHDSCP,FP=DISHDSCP.PTH)
```

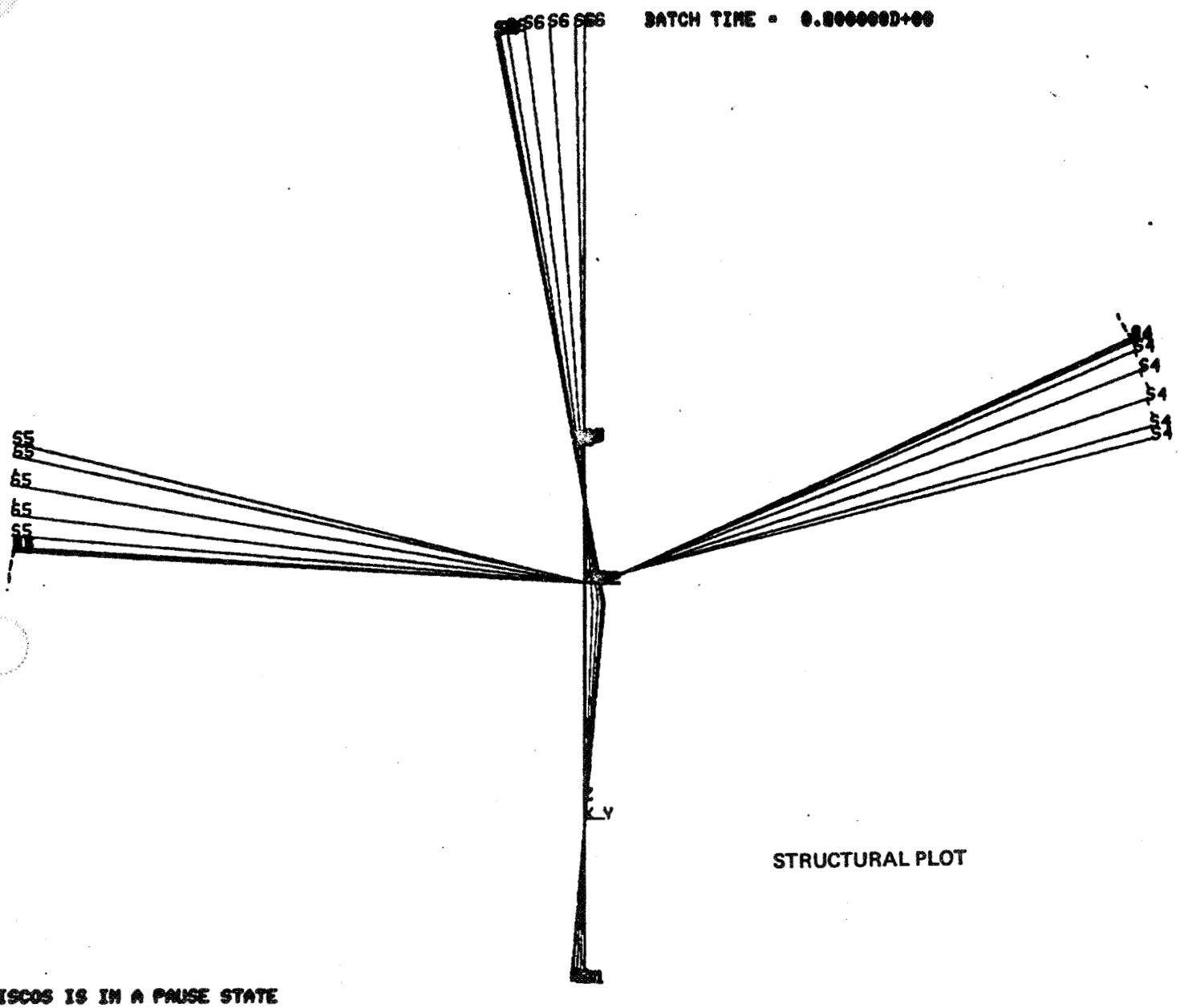
Figure C.3-2: DISCOS Typical Graphics Session

```
REAL TIME =  5.000000E+00

MENU LEVEL= 0.

 1  PICTURE DEFINITION =NONE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =CONTINUE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
1

MENU LEVEL= 0. 1.

 1  PICTURE DEFINITION =NONE
 2  PICTURE DEFINITION =STRCTURE
 3  PICTURE DEFINITION =XYCURVE

MENU ITEM NUMBER=
2

MENU LEVEL= 0. 1. 2.

 1  BODIES =ALL
 2  TIME POINTS =   ALL
 3  VIEW AXES(R,S,T) =  +X      +Y      +Z
 4  VIEW ANGLES(RR,RS,RT) =  0.,0.,0.
 5  RIGID DISPLACEMET MAXIMUM = ACTUAL
 6  RIGID DISPLACEMET FACTORS(X,Y,Z) =   1.,1.,1.
 7  FLEXIBLE DISPLACEMENT MAXIMUM = ACTUAL
 8  FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =   1.,1.,1.
 9  SCALE = UNIFORM
10  INERTIAL COORDINATE AXES =  NO
11  BODY COORDINATE AXES =  NO
12  LABELS =NONE
13  MAXIMUM X,Y,Z = ACTUAL
14  MINIMUM X,Y,Z = ACTUAL
15  RETURN

MENU ITEM NUMBER=
10

MENU LEVEL= 0. 1. 2.10.

 1  INERTIAL COORDINATE AXES =  NO
 2  INERTIAL COORDINATE AXES =  YES

MENU ITEM NUMBER=
2

MENU LEVEL= 0. 1. 2.

 1  BODIES =ALL
 2  TIME POINTS =   ALL
 3  VIEW AXES(R,S,T) =  +X      +Y      +Z
 4  VIEW ANGLES(RR,RS,RT) =  0.,0.,0.
 5  RIGID DISPLACEMET MAXIMUM = ACTUAL
 6  RIGID DISPLACEMET FACTORS(X,Y,Z) =   1.,1.,1.
 7  FLEXIBLE DISPLACEMENT MAXIMUM = ACTUAL
 8  FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =   1.,1.,1.
 9  SCALE = UNIFORM
10  INERTIAL COORDINATE AXES =  YES
11  BODY COORDINATE AXES =  NO
12  LABELS =NONE
13  MAXIMUM X,Y,Z = ACTUAL
14  MINIMUM X,Y,Z = ACTUAL
15  RETURN

MENU ITEM NUMBER=
12

MENU LEVEL= 0. 1. 2.12.

 1  LABELS =NONE
 2  LABELS =HINGES
 3  LABELS =SENSORS
 4  LABELS =BODIES
 5  LABELS =ALL

MENU ITEM NUMBER=
3

MENU LEVEL= 0. 1. 2.

 1  BODIES =ALL
 2  TIME POINTS =   ALL
 3  VIEW AXES(R,S,T) =  +X      +Y      +Z
 4  VIEW ANGLES(RR,RS,RT) =  0.,0.,0.
 5  RIGID DISPLACEMET MAXIMUM = ACTUAL
 6  RIGID DISPLACEMET FACTORS(X,Y,Z) =   1.,1.,1.
 7  FLEXIBLE DISPLACEMENT MAXIMUM = ACTUAL
 8  FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =   1.,1.,1.
 9  SCALE = UNIFORM
10  INERTIAL COORDINATE AXES =  YES
11  BODY COORDINATE AXES =  NO
12  LABELS =SENSORS
13  MAXIMUM X,Y,Z = ACTUAL
14  MINIMUM X,Y,Z = ACTUAL
15  RETURN

MENU ITEM NUMBER=
8
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)

```
MENU LEVEL= 0. 1. 2. 8.

FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =
10.,10.,10.

MENU LEVEL= 0. 1. 2.

 1  BODIES =ALL
 2  TIME POINTS =   ALL
 3  VIEW AXES(R,S,T) =  +X      +Y      +Z
 4  VIEW ANGLES(RR,RS,RT) =  0.,0.,0.
 5  RIGID DISPLACEMET MAXIMUM = ACTUAL
 6  RIGID DISPLACEMET FACTORS(X,Y,Z) =   1.,1.,1.
 7  FLEXIBLE DISPLACEMENT MAXIMUM = ACTUAL
 8  FLEXIBLE DISPLACEMENT FACTORS(X,Y,Z) =  10.,10.,10.
 9  SCALE = UNIFORM
10  INERTIAL COORDINATE AXES =  YES
11  BODY COORDINATE AXES =  NO
12  LABELS =SENSORS
13  MAXIMUM X,Y,Z = ACTUAL
14  MINIMUM X,Y,Z = ACTUAL
15  RETURN

MENU ITEM NUMBER=
15

MENU LEVEL= 0.

 1  PICTURE DEFINITION =STRCTURE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =CONTINUE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
4

MENU LEVEL= 0. 4.

 1  ACTION  AFTER EACH PICTURE =CONTINUE
 2  ACTION  AFTER EACH PICTURE =PAUSE

MENU ITEM NUMBER=
2

MENU LEVEL= 0.

 1  PICTURE DEFINITION =STRCTURE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =PAUSE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
5
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)



DISCOS IS IN A PAUSE STATE
C

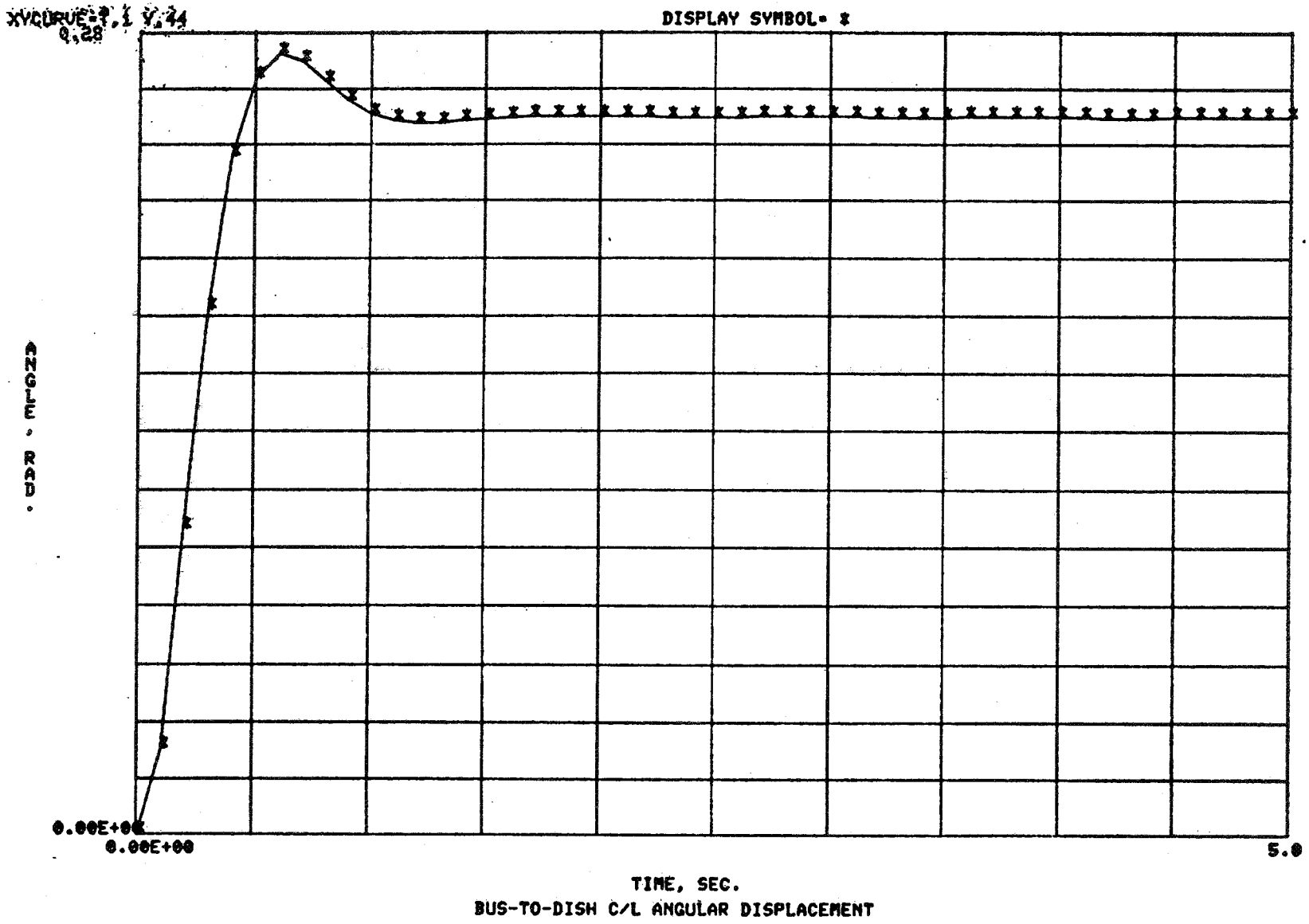
Figure C.3-2: DISCOS Typical Graphics Session (continued)

```
MENU LEVEL= 0.

 1  PICTURE DEFINITION =STRCTURE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =PAUSE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
1

MENU LEVEL= 0. 1.

 1  PICTURE DEFINITION =NONE
 2  PICTURE DEFINITION =STRCTURE
 3  PICTURE DEFINITION =XYCURVE

MENU ITEM NUMBER=
3

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 T,1
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= NONE
 7  YTITLE= NONE
 8  DISPLAY TITLE = NONE
 9  DISPLAY SYMBOL= NONE
10  RETURN

MENU ITEM NUMBER=
1

MENU LEVEL= 0. 1. 3. 1.

XYCURVE=
T,1 Y,44

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,44
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= NONE
 7  YTITLE= NONE
 8  DISPLAY TITLE = NONE
 9  DISPLAY SYMBOL= NONE
10  RETURN

MENU ITEM NUMBER=
6

MENU LEVEL= 0. 1. 3. 6.

 1  XTITLE= NONE
 2  XTITLE= USER

MENU ITEM NUMBER=
2
XTITLE=
TIME, SEC.

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,44
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= NONE
 8  DISPLAY TITLE = NONE
 9  DISPLAY SYMBOL= NONE
10  RETURN

MENU ITEM NUMBER=
7

MENU LEVEL= 0. 1. 3. 7.

 1  YTITLE= NONE
 2  YTITLE= USER

MENU ITEM NUMBER=
2
YTITLE=
ANGLE, RAD.
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)

```
MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,44
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= ANGLE, RAD.
 8  DISPLAY TITLE = NONE
 9  DISPLAY SYMBOL= NONE
10  RETURN

MENU ITEM NUMBER=
8

MENU LEVEL= 0. 1. 3. 8.

 1  DISPLAY TITLE = NONE
 2  DISPLAY TITLE = USER

MENU ITEM NUMBER=
2
DISPLAY TITLE =
BUS-TO-DISH C/L ANGULAR DISPLACEMENT

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,44
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= ANGLE, RAD.
 8  DISPLAY TITLE = BUS-TO-DISH C/L ANGULAR DISPLACEMENT
 9  DISPLAY SYMBOL= NONE
10  RETURN

MENU ITEM NUMBER=
9

MENU LEVEL= 0. 1. 3. 9.

 1  DISPLAY SYMBOL= NONE
 2  DISPLAY SYMBOL= USER

MENU ITEM NUMBER=
2

DISPLAY SYMBOL=
*

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,44
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= ANGLE, RAD.
 8  DISPLAY TITLE = BUS-TO-DISH C/L ANGULAR DISPLACEMENT
 9  DISPLAY SYMBOL= *
10  RETURN

MENU ITEM NUMBER=
10

MENU LEVEL= 0.

 1  PICTURE DEFINITION =XYCURVE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =PAUSE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
5
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)



DISCOS IS IN A PAUSE STATE

Figure C.3-2: DISCOS Typical Graphics Session (continued)

```
MENU LEVEL= 0.

 1  PICTURE DEFINITION =XYCURVE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =PAUSE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
1

MENU LEVEL= 0. 1.

 1  PICTURE DEFINITION =NONE
 2  PICTURE DEFINITION =STRCTURE
 3  PICTURE DEFINITION =XYCURVE

MENU ITEM NUMBER=
3

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,44
 2  XMAX  = ACTUAL
 3  XMIN  = ACTUAL
 4  YMAX  = ACTUAL
 5  YMIN  = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= ANGLE, RAD.
 8  DISPLAY TITLE = BUS-TO-DISH C/L ANGULAR DISPLACEMENT
 9  DISPLAY SYMBOL= *
10  RETURN

MENU ITEM NUMBER=
1

MENU LEVEL= 0. 1. 3. 1.

XYCURVE=
T,1 Y,25 Y,26,Y,27 Y,28 Y,29 Y,30
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)

```
BAD INPUT

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,25 Y,26,Y,27 Y,28 Y,29 Y,30
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= ANGLE, RAD.
 8  DISPLAY TITLE = BUS-TO-DISH C/L ANGULAR DISPLACEMENT
 9  DISPLAY SYMBOL= X
10  RETURN

MENU ITEM NUMBER=
7

MENU LEVEL= 0. 1. 3. 7.

 1  YTITLE= NONE
 2  YTITLE= USER

MENU ITEM NUMBER=
2
YTITLE=
MODAL RESPONSE, IN.

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,25 Y,26,Y,27 Y,28 Y,29 Y,30
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= MODAL RESPONSE, IN.
 8  DISPLAY TITLE = BUS-TO-DISH C/L ANGULAR DISPLACEMENT
 9  DISPLAY SYMBOL= X
10  RETURN

MENU ITEM NUMBER=
8

MENU LEVEL= 0. 1. 3. 8.

 1  DISPLAY TITLE = NONE
 2  DISPLAY TITLE = USER

MENU ITEM NUMBER=
2
DISPLAY TITLE =
MODAL DISPLACEMENT FOR MODES 2,3,5,6,7 & 8

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,25 Y,26,Y,27 Y,28 Y,29 Y,30
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= MODAL RESPONSE, IN.
 8  DISPLAY TITLE = MODAL DISPLACEMENT FOR MODES 2,3,5,6,7 &
 9  DISPLAY SYMBOL= X
10  RETURN

MENU ITEM NUMBER=
9

MENU LEVEL= 0. 1. 3. 9.

 1  DISPLAY SYMBOL= NONE
 2  DISPLAY SYMBOL= USER

MENU ITEM NUMBER=
2
DISPLAY SYMBOL=
2,3,5,6,7,8

MENU LEVEL= 0. 1. 3.

 1  XYCURVE=T,1 Y,25 Y,26,Y,27 Y,28 Y,29 Y,30
 2  XMAX   = ACTUAL
 3  XMIN   = ACTUAL
 4  YMAX   = ACTUAL
 5  YMIN   = ACTUAL
 6  XTITLE= TIME, SEC.
 7  YTITLE= MODAL RESPONSE, IN.
 8  DISPLAY TITLE = MODAL DISPLACEMENT FOR MODES 2,3,5,6,7 &
 9  DISPLAY SYMBOL= 2,3,5,6,7,8
10  RETURN

MENU ITEM NUMBER=
10

MENU LEVEL= 0

 1  PICTURE DEFINITION =XYCURVE
 2  PICTURE MODE = BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =PAUSE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
5
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)

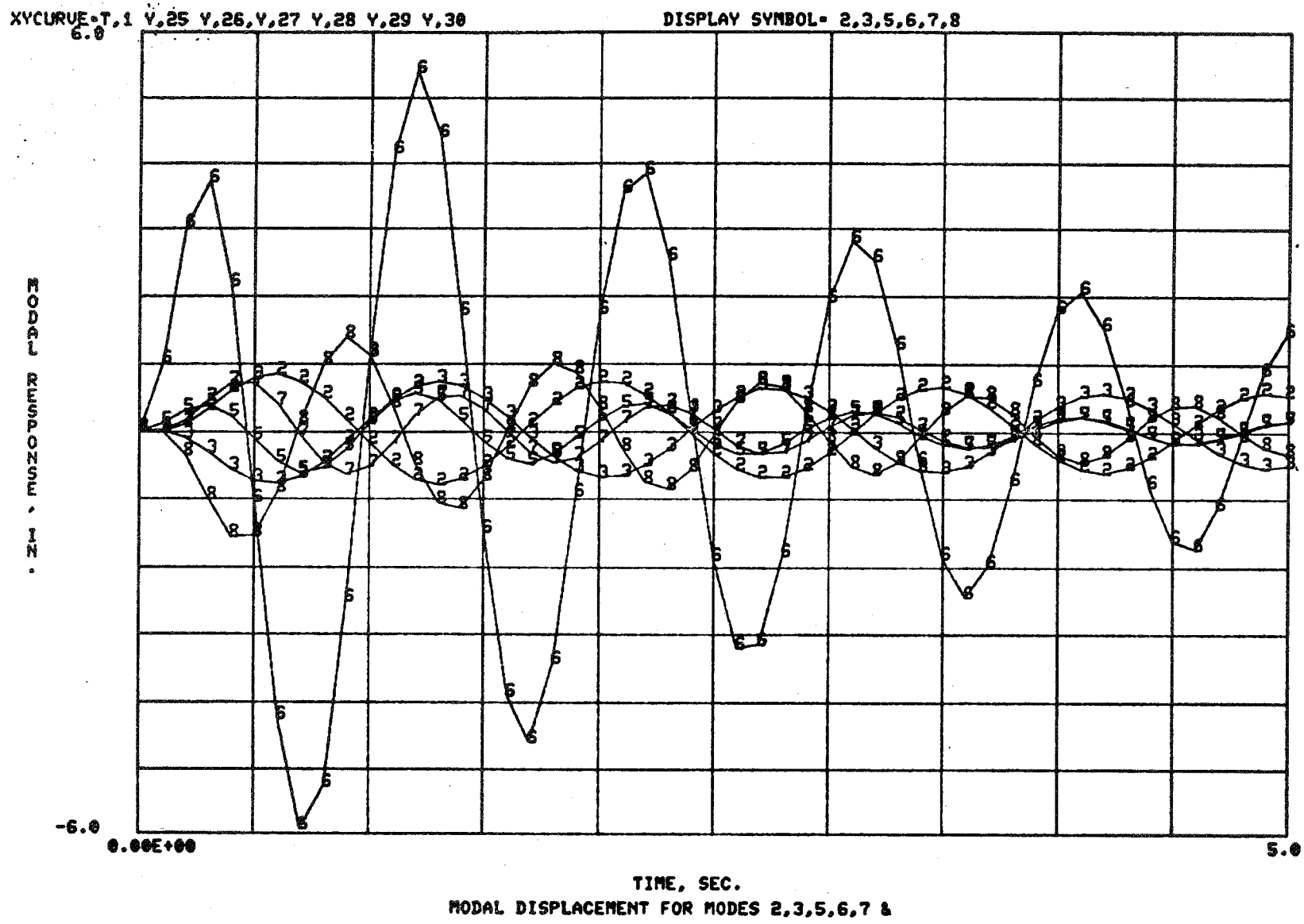


Figure C.3-2: DISCOS Typical Graphics Session (continued)

DISCOS IS IN A PAUSE STATE

```
MENU LEVEL= 0.

 1  PICTURE DEFINITION =XYCURVE
 2  PICTURE MODE =  BATCH
 3  SCREEN ERASURE =MANUAL
 4  ACTION  AFTER EACH PICTURE =PAUSE
 5  DISPLAY PICTURE
 6  ALTER INPUT
 7  INTEGRATE TO NEXT DISPLAY POINT
 8  STOP

MENU ITEM NUMBER=
8
```

Figure C.3-2: DISCOS Typical Graphics Session (continued)

APPENDIX D - ORACLS IMPLEMENTATION

The ORACLS program has been implemented within IAC so as to allow execution in either of two modes - standard or user-defined. Section D.1 discusses the two standard ORACLS driver programs, REGEST (full state regulator and estimator design) and TIMFST (time response of open and closed loop systems). Their input/output formats and example problems are presented. Section D.2 discusses the general methodology for developing and executing user-defined ORACLS drivers within IAC. New IAC ORACLS utility procedures available for use by the user programmer are described in Section D.3. Section D.4 discusses modification and operation of the original ORACLS software package, associated with the VAX conversion and IAC implementation.

D.1 STANDARD DRIVERS

Two standard ORACLS drivers have been written. The first (REGEST) solves the linear optimal regulator/estimator/compensator problem for either continuous or discrete systems. The second standard driver (TIMFST) calculates a time response for a closed or open loop system.

The forms of the IAC RUN statements for these drivers are:

```
RUN ORACLS (STD=REGEST, F=filename, FIN=filespec, AREA=code, PRT=option)
RUN ORACLS (STD=TIMFST, F=filename, FIN=filespec, AREA=code, PRT=option)
```

The STD parameter names the standard driver (default is REGEST). The F parameter names the input file (filename.DAT) and the output file (filename.LIS). FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec filename.DAT. The optional AREA parameter gives the storage area (H, D, HD or DH) from/to which IAC arrays are to be transferred. The optional PRT parameter specifies disposition (YES, NO or HOLD) of the output file.

D.1.1 REGEST

REGEST is an interactive program that calculates a linear optimal regulator, estimator, and/or compensator. A user defined input file is required which specifies IAC arrays and defines regulator and estimator options. Interactive prompts are given to allow the user to select output to be printed or stored, change the weighting matrices, add an alpha shift, or create plot files to be used by the IAC PLOT module.

Topics in this section describe the input file rules and data; an input file example; the output; and an example REGEST run.

Input File Rules - The input file is arranged in blocks, one block each for the regulator, estimator, and compensator. The following general rules apply.

1) Any or all blocks may appear in any order. Note, however, that all blocks needed for the user's problem must appear.
2) Each block begins with a header line consisting of a single letter in column 1: R (for regulator), E (for estimator), C (for compensator).
3) All data lines for the regulator, estimator or compensator block must immediately follow the header line. Data lines within a block may appear in any order.
4) Each data line consists of three items: (a) a nametag starting in column 1; (b) an "=" and/or one or more adjacent blanks; (c) a maximum 80-character value, not beginning with an "=" or a blank.
5) A comment line is any line which is blank or which begins with a "$" in column 1. Comment lines may occur anywhere within the input file. They are ignored in data processing, except that they are included in the count of line numbers.

Input File Data - The data lines within each block are defined below. Note that "L" represents a logical value T (true) or F (false), and "X" represents a double precision real value. The ALPHA value is not an alpha shift. (An alpha shift may be defined interactively by the user). Note also that text in parentheses is provided as information only and is not to be included in the input file.

For regulator or estimator block:

R (reg) or E (est)
DISC = L
NEWT = L
STABLE = L (required if NEWT is true)
FNULL = L (required if NEWT is true)
ALPHA = X (required if NEWT is true and STABLE is false)
A = matrix A spec
B = matrix B (reg) or C = matrix C (est) spec
Q = matrix Q (reg) or QE = matrix QE (est) spec
R = matrix R (reg) or RE = matrix RE (est) spec
F = matrix F (reg) or G = matrix G (est) spec (required if NEWT is true and A is not asymptotically stable)
IDENT = L
H = matrix H (reg) or GG = matrix GG (est) (required if IDENT is false)
P = matrix P (reg) or PE = matrix PE (est) (required if NEWT is false)
FOUT = output matrix F (reg) or GOUT = output matrix G (est) spec

For compensator block:

C
A = matrix A spec
B = matrix B spec
C = matrix C spec
F = matrix F spec
G = matrix G spec

A, B, and C are IAC array specifications for the system matrices. F and G are system gain matrices (output for the regulator and estimator, input for the compensator). The user has the choice to store these matrices at the end of each regulator and estimator run. When the compensator is run the last gain matrices stored will be the ones used. For the regulator, Q is the state weighting matrix and R is the control weighting matrix. For the estimator, QE is the process noise weighting matrix and RE is the measurement noise weighting matrix.

See subroutines ASYMREG (reg) and ASYMFIL (est) in the ORACLS manual, Reference 13, for information on the definition of the remaining matrices and logical variables.

Input File Example - An example of an actual input file is shown in Figure D.1-1.

```
$ REGEST EXAMPLE INPUT FILE
E
DISC = F
NEWT = F
A = AO.DAT
C = CO.DAT
QE = QO.DAT
RE = RE.DAT
IDENT = T
PE = PO.DAT
GOUT = GOPTEST.DAT
R
DISC = F
NEWT = F
A = AO.DAT
B = BO.DAT
Q = QO.DAT
R = RO.DAT
IDENT = T
P = PO.DAT
FOUT = FOPTREG.DAT
C
A = AO.DAT
B = BO.DAT
C = CO.DAT
F = FOPTREG.DAT
G = GOPTEST.DAT
```

Figure D.1-1: Example REGEST Input File

Output - At the beginning of REGEST certain program options are set, some of which relate to the output. At the beginning of execution of each input block the user is prompted about changing these options. See the ORACLS manual subroutines ASYMFIL and ASYMREG for a description of options.

Input matrices A, B, C, Q/QE, R/RE, H, and P/PE along with the eigenvalues of the system are written to the filename.LIS file. During REGEST execution the eigenvalues are also typed to the screen. The user is then prompted to determine whether output matrices of the regulator or estimator are to be stored.

During execution of each input block the user is prompted about creating plot files. If the user desires, plot files of the eigenvalues will be written to the host directory, with a spec which the user selects. An example plot made from a plot file is shown in Figure D.1-2.

Example REGEST Run - An example run utilizing REGEST is presented to illustrate the use of REGEST within IAC. The user creates the input data file EXP1.DAT, the contents of which are shown in Figure D.1-1. FOPTREG.DAT and GOPTEST.DAT define output array specs to be used by the regulator and the estimator, respectively; they define input array specs to be used by the compensator. The RUN statement for this example is

```
RUN ORACLS (STD = REGEST, F = EXP1)
```

See Figure D.1-2 for an example of an interactive display of the REGEST run. The resulting output file is shown in Figure D.1-3.

```
ACE >RUN ORACLS(STD=REGEST,F=EXP1)
BEGIN ORACLS      RUNID=12425285

 WHAT TITLE WOULD YOU LIKE FOR THIS RUN?
EXAMPLE REGEST RUN




EXAMPLE REGEST RUN
 ORACLS  PROGRAM


 PROGRAM OPTIONS START AT:

      *GAIN MATRICES AND SOLUTION TO RICCATI
       EQUATION WILL BE PRINTED TO OUTPUT FILE

      *INTERMEDIATE VALUES WILL NOT BE PRINTED

      *ONLY STEADY STATE VALUES WILL BE COMPUTED

      *RICCATI RESIDUAL WILL NOT BE PRINTED

      *MATRICES A-BF AND A-GC AND THEIR EIGENVALUES
       WILL NOT BE PRINTED BY ASYREG OR ASYFIL

 THE USER WILL HAVE AN OPPORTUNITY TO CHANGE
 THESE OPTIONS WHEN EITHER THE REGULATOR OR
 THE ESTIMATOR IS RUN


 DO YOU WANT TO START WITH REGULATOR, ESTIMATOR, COMPENSATOR
 OR RUN IN ONE STEP?
 ENTER R-(REG) E-(EST) C-(COMP) O-(ONE STEP)
R
```

Figure D.1-2: Example REGEST Run - Interactive Display

```
-----------------------------------------------

 PROGRAM TO SOLVE THE CONTINUOUS OR DISCRETE TIME INVARIANT ASYMPTOTIC
 LINEAR OPTIMAL OUTPUT REGULATOR PROBLEM WITH NOISE FREE MEASUREMENTS

-----------------------------------------------

*INFORMATION* KSC303:[RGV.ORACLS.REGHD]AO.DAT;7 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]BO.DAT;6 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]QO.DAT;6 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]RO.DAT;6 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]PO.DAT;6 GET COMPLETED

 WHAT CASE NUMBER DO YOU WANT TO USE TO START?

1

DO YOU WANT TO CHANGE PROGRAM OPTIONS? N

 ALPHA SHIFT IS:
 0.0000000000000000E+00


DO YOU WANT TO CHANGE THE ALPHA SHIFT? Y
 ENTER ALPHA
.1

         -REGULATOR CALCULATION IN PROGRESS-

              .
              .
              .
```

Figure D.1-2: Example REGEST Run - Interactive Display (Continued)

```
                                CASE NO.      1

                    EIGENVALUES OF FULL ORDER REGULATOR  [A - BF]

              EIGENVALUES
       REAL                  IMAGINARY           DAMP RATIO          FREQUENCY
     -2.39784805              1.36310692          0.86934858          2.75821242
     -2.39784805             -1.36310692          0.86934858          2.75821242
     -3.05348133              3.08539936          0.70342080          4.34090284
     -3.05348133             -3.08539936          0.70342080          4.34090284
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]REGEV.DAT;33 PUT COMPLETED


DO YOU WANT TO CREATE A PLOT FILE? Y
 WHAT SPEC DO YOU WANT FOR THE PLOT FILE?
REG.PLT

KSC303:[RGV.ORACLS.REGHD]REG.PLT;3  PLOT FILE WRITTEN


DO YOU WANT TO LOOK AT PLOT? Y

BEGIN ACE RUNID=12481778
Enter command, or EXIT to terminate, or %H for help.
ACE >RUN PLOT

                    •
                    •
                    •
```

Figure D.1-2: Example REGEST Run - Interactive Display (Continued)

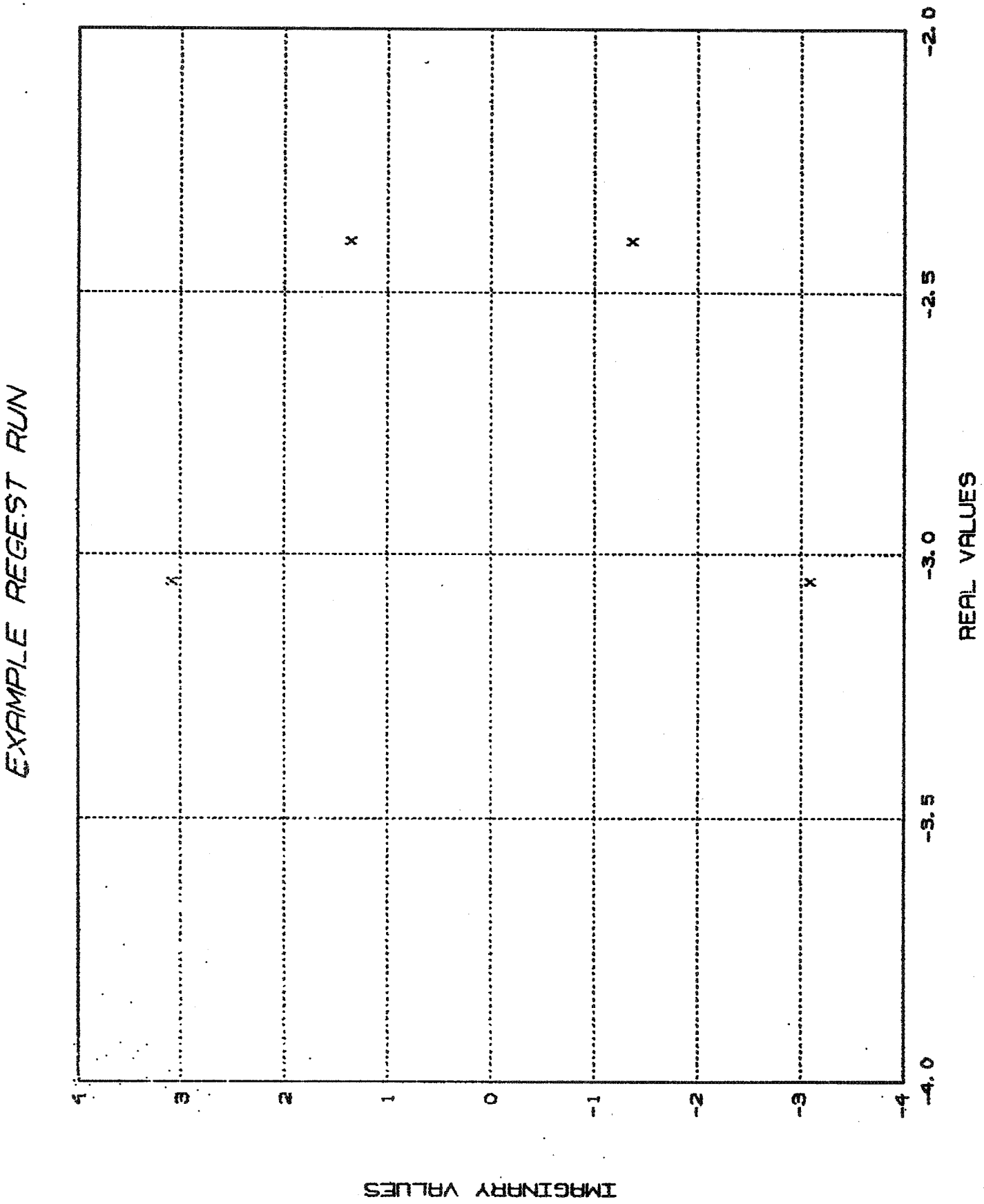


Figure D.1-2: Example REGEST Run - Interactive Display (Continued)

```
FORTRAN STOP
END ACE

DO YOU WANT TO STORE GAIN MATRIX F? Y
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]FOPTREG.DAT;25 PUT COMPLETED

DO YOU WANT TO RUN THE REGULATOR AGAIN? N

 REGULATOR DESIGN HAS BEEN COMPLETED

 DO YOU WANT TO GO ON TO ESTIMATOR
 COMPENSATOR OR STOP?
 ENTER E-(EST) C-(COMP) S-(STOP)
C

-----------------------------------------------

           COMPENSATOR CALCULATION

-----------------------------------------------


 WHAT CASE NUMBER DO YOU WANT TO USE TO START?

1
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]AO.DAT;7 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]BO.DAT;6 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]CO.DAT;6 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]FOPTREG.DAT;25 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]GOPTEST.DAT;23 GET COMPLETED
 NOTE: A GAIN MATRIX G HAS NOT BEEN STORED
       THIS RUN, A PREVIOUS G WILL BE USED


    -COMPENSATOR CALCULATION IN PROGRESS-
```

Figure D.1-2: Example REGEST Run - Interactive Display (Continued)

```
                                CASE NO.      1

                    EIGENVALUES OF FULL ORDER COMPENSATOR  [A - GC - BF]

              EIGENVALUES
        REAL                  IMAGINARY          DAMP RATIO          FREQUENCY
       2.64424171             0.00000000        -1.00000000          2.64424171
     -13.94724686             0.00000000         1.00000000         13.94724686
     -21.06168574             8.19785460         0.93189703         22.60087225
     -21.06168574            -8.19785460         0.93189703         22.60087225

DO YOU WANT TO STORE COMPENSATOR DATA? Y
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]COMEV.DAT;19 PUT COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.REGHD]COMP.DAT;17 PUT COMPLETED

DO YOU WANT TO CREATE A PLOT FILE? N

 COMPENSATOR COMPUTATION HAS BEEN COMPLETED

 DO YOU WANT TO RUN REGULATOR, ESTIMATOR
 OR STOP
 ENTER R-(REG),E-(EST),S-(STOP)
S
*INFORMATION* CONDITIONS DETECTED DURING THE RUN
                    WARNING  ERROR  ABORT
                          0      0      0
FORTRAN STOP
END ORACLS
*INFORMATION* LAST ACE COMMAND PROCESSED
```

Figure D.1-2: Example REGEST Run - Interactive Display (Continued)

```
**********************************************
**********************************************
**********************************************
**********      REGULATOR      **********
**********************************************
**********************************************
**********************************************


**********************************************
**********      CASE NO.     1     **********
**********************************************
ALPHA SHIFT IS:   0.100E+00


PROGRAM TO SOLVE THE TIME-INVARIANT INFINITE-DURATION CONTINUOUS OPTIMAL
REGULATOR PROBLEM  WITH NOISE-FREE MEASUREMENTS

     A    MATRIX       4 ROWS       4 COLUMNS
-2.5000000D+00   2.5000000D-01  -3.8000000D+01   0.0000000D+00
-7.5000000D-02  -1.7000000D-01   4.4000000D+00   0.0000000D+00
 7.8000000D-02  -9.9000000D-01  -1.3000000D-01   5.2000000D-02
 1.0000000D+00   7.8000000D-02   0.0000000D+00   1.0000000D-01

     B    MATRIX       4 ROWS       2 COLUMNS
 1.7000000D+01   7.0000000D+00
 2.0000000D-02  -3.2000000D+00
 0.0000000D+00   4.6000000D-02
 0.0000000D+00   0.0000000D+00

     Q    MATRIX       4 ROWS       4 COLUMNS
 1.0000000D+00   0.0000000D+00   0.0000000D+00   0.0000000D+00
 0.0000000D+00   1.0000000D+02   0.0000000D+00   0.0000000D+00
 0.0000000D+00   0.0000000D+00   1.0000000D+02   0.0000000D+00
 0.0000000D+00   0.0000000D+00   0.0000000D+00   1.0000000D+02

H IS AN IDENTITY MATRIX


     R    MATRIX       2 ROWS       2 COLUMNS
 1.0000000D+02   0.0000000D+00
 0.0000000D+00   1.0000000D+02

                    .
                    .
                    .
```

Figure D.1-3: Example REGEST Run - Output File

```
      P    MATRIX       4 ROWS       4 COLUMNS

                    .
                    .
                    .

EXAMPLE REGEST RUN
 ORACLS  PROGRAM

  1.2825286D+00   2.5067135D+00  -8.8685415D+00   5.9753687D+00
  2.5067135D+00   4.3901612D+01  -4.3812344D+01   1.2655661D+01
 -8.8685415D+00  -4.3812344D+01   2.8843201D+02  -4.4038683D+01
  5.9753687D+00   1.2655661D+01  -4.4038683D+01   4.1436413D+01

     F    MATRIX       2 ROWS       4 COLUMNS
  2.1853121D-01   4.3492161D-01  -1.5164145D+00   1.0183438D+00
  5.4826427D-03  -1.2495353D+00   9.1387583D-01  -6.9631272D-03

                EIGENVALUES OF FULL ORDER REGULATOR  [A - BF]

           EIGENVALUES
      REAL              IMAGINARY         DAMP RATIO         FREQUENCY
    -2.39784805          1.36310692         0.86934858         2.75821242
    -2.39784805         -1.36310692         0.86934858         2.75821242
    -3.05348133          3.08539936         0.70342080         4.34090284
    -3.05348133         -3.08539936         0.70342080         4.34090284
************************************************
************************************************
************************************************
**************     COMPENSATOR   ***************
**************    CASE NO.     1    ***************
************************************************
************************************************
************************************************

NOTE:  REGULATOR CASE NO.     1 AND AN ESTIMATOR FROM A PREVIOUS RUN
       WERE USED TO COMPUTE COMPENSATOR.

                EIGENVALUES OF FULL ORDER COMPENSATOR  [A - GC - BF]

           EIGENVALUES
      REAL              IMAGINARY         DAMP RATIO         FREQUENCY
     2.64424171          0.00000000        -1.00000000         2.64424171
   -13.94724686          0.00000000         1.00000000        13.94724686
   -21.06168574          8.19785460         0.93189703        22.60087225
   -21.06168574         -8.19785460         0.93189703        22.60087225
```

Figure D.1-3: Example REGEST Run - Output File (Continued)

D.1.2 TIMFST

TIMFST is an interactive program, which computes a time response for an open or closed loop system. TIMFST puts either the open or closed loop system matrices into modal coordinates. This results in sparse matrices allowing item by item time response computation for efficient computer operation. A user defined input file is required which references IAC input arrays. Interactive queries allow the user to choose whether a debug option is to be set, what type of system is to be analyzed, and which states are to be printed or plotted.

Topics in this section describe the input file rules and data; an input file example; the block diagram and matrix definitions; the output; and an example TIMFST run.

Input File Rules - The following general rules apply.

1) Data lines may appear in any order.
2) Each data line consists of three items: (a) a nametag starting in column 1; (b) an "=" and/or one or more adjacent blanks; (c) a maximum 80-character value, not beginning with an "=" or a blank.
3) A comment line is any line which is blank or which begins with a "$" in column 1. Comment lines may occur anywhere within the input file. They are ignored in data processing, except that they are included in the count of line numbers.

Input File Data - The data lines are defined below. Note that text in parentheses is provided as information only and is not to be included in the input file. The IPRINT and IPLOT values are independent of one another. The XI(n,1) matrix has an arbitrary size n; for n > number of states, XI values are ignored; for state numbers > n, zero values are assumed.

TSTEP = time step (to be used in calculation)
TFINAL = final time
IPRINT = print interval (integer multiplier on time step, default=0 indicates no print times)
IPLOT = plot interval (integer multiplier on time step, default=0 indicates no plot times)

A = matrix A spec (plant matrix)
B = matrix B spec (plant matrix)
C = matrix C spec (plant matrix)
XI = matrix XI spec (initial conditions for all states, default = all zero values)
D = matrix D spec (optional - plant direct feed through)
C2 = matrix C2 spec (required if non-measured outputs)
J = matrix J spec (required if system is closed loop)
E = matrix E spec (required if compensator)
F = matrix F spec (required if compensator)
G = matrix G spec (required if compensator)
H = matrix H spec (required if compensator)

A, B and C are IAC array specifications for the plant definition matrices. D is the plant direct feed through matrix, C2 relates non-measured outputs to plant states, and J is an output feedback matrix. E is an estimator dynamic matrix, F is an regulator gain matrix, G is an estimator gain matrix, and H specifies the dependency of the compensator on the control.

Input File Example - An example of an actual input file is shown in Figure D.1-4.

```
$ TIMFST EXAMPLE INPUT FILE
A = A.DAT
B = B.DAT
C = C.DAT
XI = INITIAL.DAT
TSTEP = .5
TFINAL = 50.
IPRINT = 1
IPLOT = 1
D = PDFT.DAT
C2 = MAT.C2
J = J.DAT
E = ECOMP.DAT
F = FCOMP.DAT
G = GCOMP.DAT
H = HCOMP.DAT
```

Figure D.1-4: Example TIMFST Input File

Equations, Block Diagram, and Matrix Definitions - The equations used are defined in Figure D.1-5 and a block diagram is shown in Figure D.1-6. The E, G and H matrices are defined in Figure D.1-7.

Output - All input arrays are echoed out to the output listing file.

The user will be able to set a DEBUG option through an interactive prompt. If the DEBUG option is set several matrices used during computation will be printed on the output listing file. Included are the matrices shown in Figure D.1-8.

The user may select the states and outputs which are to be printed. The plant states are first, followed by estimated states. The outputs are ordered such that plant outputs are first, followed by controller outputs and other (non-measured) outputs.

The user may also select certain plot files to be written to the host default directory. The spec of each plot file is STATEn.PLT or OUTPUTn.PLT, where n is the number of the state or output selected.

Example TIMFST Run - An example run utilizing TIMFST is presented to show the use of TIMFST within IAC. The user creates the input data file EXP2.DAT, the contents of which are shown in Figure D.1-4. The RUN statement for this example is

```
RUN ORACLS (STD = TIMFST, F = EXP2)
```

See Figure D.1-9 for an example of an interactive display of the TIMFST run. The resulting output file is shown in Figure D.1-10, and an associated plot is shown in Figure D.1-11.

Original Equations:

$$\dot{x} = Ax + Bu \qquad y = Cx + Du$$

Closed Loop Expands To:

$$\begin{Bmatrix} \dot{x} \\ \dot{\xi} \end{Bmatrix} = \begin{bmatrix} Anew \end{bmatrix} \begin{Bmatrix} x \\ \xi \end{Bmatrix} + \begin{bmatrix} Bnew \end{bmatrix} \begin{Bmatrix} u_e \\ \theta \end{Bmatrix}$$

$$\begin{Bmatrix} y_1/\sigma/y_2 \end{Bmatrix} = \begin{bmatrix} Cnew \end{bmatrix} \begin{Bmatrix} x \\ \xi \end{Bmatrix} + \begin{bmatrix} Dnew \end{bmatrix} \begin{Bmatrix} u_e \\ \theta \end{Bmatrix}$$

Where

- x = plant states
- ξ = estimated states
- y_1 = plant outputs
- σ = controller outputs
- y_2 = other, e.g. nonmeasured outputs
- u_e = external inputs
- θ = auxiliary inputs

and Anew, Bnew, Cnew and Dnew are as defined in Figure D.1-8.

Figure D.1-5: Equations for TIMFST

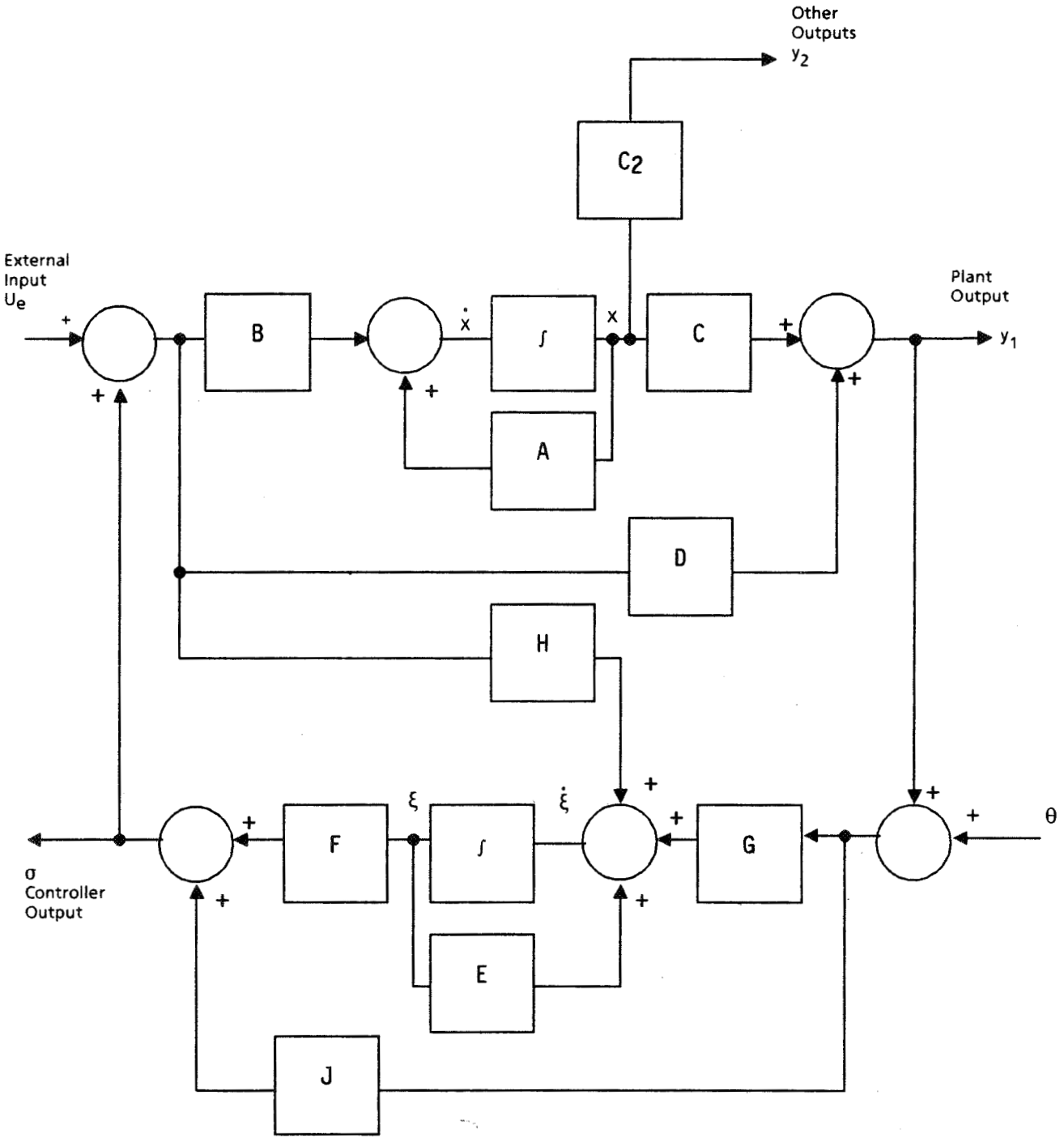


Figure D.1-6: TIMFST Block Diagram

Symbol Definitions

x = State

y = Output

u = input

ξ = Estimated State

$\hat{y}$ = Estimated Output

K = Estimator Gain matrix

A, B, C, D = Plant Matrices

Plant Equations

$$\dot{x} = Ax + Bu \ , \ y = Cx + Du$$

Estimator Equations

$$\dot{\xi} = A\xi + Bu + K(y - \hat{y})$$

$$\hat{y} = C\xi + Du$$

From Above Equations

$$\dot{\xi} = (A-KC)\xi + Ky + (B - KD)u$$

Since: $\dot{\xi} = E\xi + Gy + Hu$ (From Block Diagram)

Then: $G = K$

$E = A - GC$

$H = B - GD$

Note - If the REGEST designed compensator matrices (E, F, G) are used (where $E = A-BF-GC$), then the TIMFST $H = -GD$, the TIMFST $F = -F$, and the TIMFST u_e becomes a disturbance input through the control.

Figure D.1-7: Definitions of TIMFST Compensator Matrices

$$Anew = \left[\begin{array}{c|c} A+B\Delta JC & B\Delta F \\ \hline GC+GD\Delta JC+H\Delta JC & E+GD\Delta F+H\Delta F \end{array}\right]$$

$$Bnew = \left[\begin{array}{c|c} B\Delta & B\Delta J \\ \hline GD\Delta+H\Delta & H+\Delta J+G+GD\Delta J \end{array}\right]$$

$$Cnew = \left[\begin{array}{c|c} C+D\Delta JC & D\Delta F \\ \hline JC+JD\Delta JC & F+JD\Delta F \\ \hline C & 0 \end{array}\right]$$

$$Dnew = \left[\begin{array}{c|c} D\Delta & D\Delta J \\ \hline JD\Delta & J+JD\Delta J \\ \hline 0 & 0 \end{array}\right]$$

where $\Delta = (I - JD)^{-1}$

Figure D.1-8: TIMFST Debug Printout

Matrix	Size	Where
A	(n,n)	n = number of system states
B	(n,k)	k = number of actuators
C	(l,n)	l = number of sensors
D	(l,k)	ne = number of estimator states
Anew	(n+ne,n+ne)	l_2 = number of non-measured outputs
Bnew	(n+ne,n+ne)	
Cnew	(n+ne,k+l)	
Dnew	(l+k+l_2,k+l)	
u	(K,1) (Open Loop))	
u	(k+l,1) (Closed Loop)	
T	Same as A/Anew	

T is a matrix used to put the system into modal coordinates. The modal form used in TIMFST is

$$\begin{bmatrix} \sigma & -\omega \\ \omega & \sigma \end{bmatrix}$$

where σ and ω are respectively the real and imaginary parts of A (for open loop systems) or Anew (for closed loop systems).

Figure D.1-8: TIMFST Debug Printout (Continued)

```
ACE >RUN ORACLS(STD=TIMFST,F=EXP2)
BEGIN ORACLS      RUNID=13210687

 WHAT TITLE DO YOU WANT FOR THIS RUN?
EXAMPLE TIMFST RUN

DO YOU WANT DEBUG SET? Y
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]A.DAT;6 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]B.DAT;5 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]INITIAL.DAT;4 GET COMPLETED

 DO YOU WANT TO USE A:

      1) D MATRIX (PLANT DIRECT FEED THROUGH)

      2) C2 MATRIX (NON-MEASURED OUTPUT)

      3) BOTH OF ABOVE

      4) NONE OF ABOVE

 Enter number desired-
3
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]PDFT.DAT;4 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]MAT.C2;5 GET COMPLETED

DO YOU WANT TO RUN CLOSED LOOP? Y

 PROGRAM TO SOLVE THE CLOSED LOOP TIME RESPONSE
```

Figure D.1-9: Example TIMFST Run - Interactive Display

```
 DO YOU WANT:

      1) STATE FEEDBACK (UNIT C; INPUT J; NO E,F,G,H)

      2) OUTPUT FEEDBACK ONLY (INPUT C,J; NO E,F,G,H)

      3) A COMPENSATOR (INPUT C, E,F,G,H; INPUT OR ZERO J)

 Enter number desired-
3
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]C.DAT;4 GET COMPLETED

DO YOU WANT AN OUTPUT FEEDBACK MATRIX J? Y
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]J.DAT;4 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]ECOMP.DAT;4 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]FCOMP.DAT;4 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]GCOMP.DAT;4 GET COMPLETED
*INFORMATION* KSC303:[RGV.ORACLS.TIMHD]HCOMP.DAT;4 GET COMPLETED

 PERFORMING EIGENSOLUTION

                    EIGENVALUES OF CLOSED LOOP DYNAMICS MATRIX

              EIGENVALUES
       REAL                IMAGINARY          DAMP RATIO          FREQUENCY
    -0.01990422          0.00000000          1.00000000          0.01990422
    -0.17492237          0.00000000          1.00000000          0.17492237
    -0.14636817          0.11317448          0.79109720          0.18501919
    -0.14636817         -0.11317448          0.79109720          0.18501919
    -0.55088873          0.73164082          0.60150704          0.91584752
    -0.55088873         -0.73164082          0.60150704          0.91584752
    -0.52247781          1.43028474          0.34311982          1.52272700
    -0.52247781         -1.43028474          0.34311982          1.52272700

DO YOU WANT TO CONTINUE? Y
```

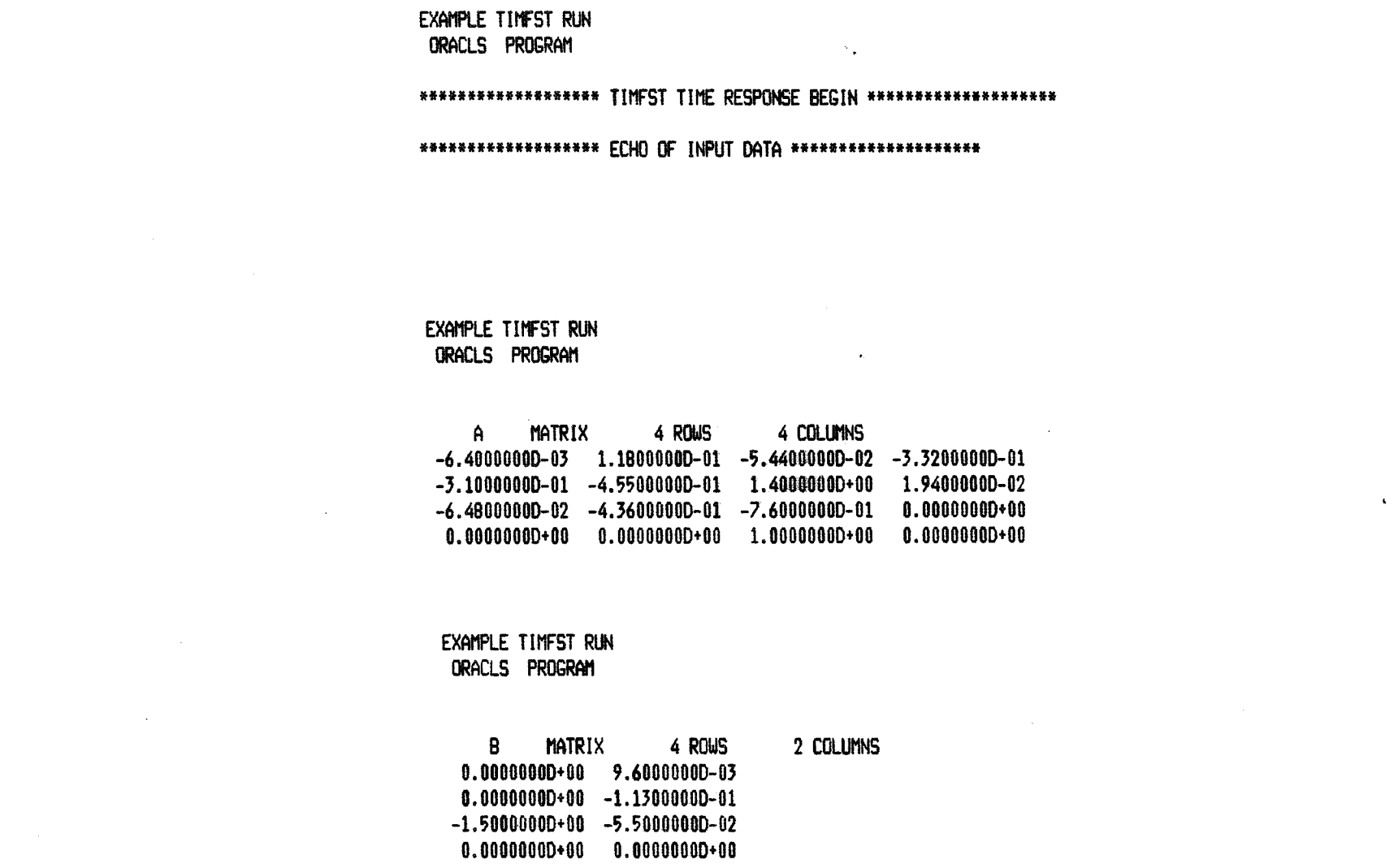
Figure D.1-9: Example TIMFST Run - Interactive Display (Continued)

```
-------------------------------------------------

Plant states are STATE 1 thru STATE   4

Estimated states are STATE   5 thru STATE   8

Plant outputs are OUTPUT 1 thru OUTPUT   4

Controller outputs are OUTPUT   5 thru OUTPUT   6

Other outputs are OUTPUT   7 thru OUTPUT   8


 For the following questions
 answer with "ALL" ,"NONE"  or
 a list of numbers (blank or comma delimiters).

 WHAT STATES DO YOU WANT TO PRINT?
ALL
 WHAT STATES DO YOU WANT TO PLOT?
2,3
 WHAT OUTPUTS DO YOU WANT TO PRINT?
ALL
 WHAT OUTPUTS DO YOU WANT TO PLOT?
NONE
```

Figure D.1-9: Example TIMFST Run - Interactive Display (Continued)

```
External inputs are INPUT 1 thru INPUT   2

Auxiliary inputs are INPUT   3 thru INPUT   6

 WHAT TYPE OF INPUT DO YOU WANT?
      1) STEP
      2) RAMP
 Enter number desired-
1

DO YOU WANT SAME STEP SIZE FOR ALL INPUTS? N

Enter step size for U(  1)-
.1
Enter step size for U(  2)-
0
Enter step size for U(  3)-
0
Enter step size for U(  4)-
0
Enter step size for U(  5)-
0
Enter step size for U(  6)-
0

 PERFORMING TIME RESPONSE


 GENERATING PLOT FILES


KSC303:[RGV.ORACLS.TIMHD]STATE2.PLT;2  PLOT FILE WRITTEN
KSC303:[RGV.ORACLS.TIMHD]STATE3.PLT;2  PLOT FILE WRITTEN
*INFORMATION* CONDITIONS DETECTED DURING THE RUN
                    WARNING  ERROR  ABORT
                          0      0      0
FORTRAN STOP
END ORACLS
*INFORMATION* LAST ACE COMMAND PROCESSED
```

Figure D.1-9: Example TIMFST Run - Interactive Display (Continued)



```
EXAMPLE TIMFST RUN
 ORACLS  PROGRAM

****************** TIMFST TIME RESPONSE BEGIN ******************

****************** ECHO OF INPUT DATA ******************
```

```
EXAMPLE TIMFST RUN
 ORACLS  PROGRAM

    A     MATRIX       4 ROWS       4 COLUMNS
-6.4000000D-03   1.1800000D-01  -5.4400000D-02  -3.3200000D-01
-3.1000000D-01  -4.5500000D-01   1.4000000D+00   1.9400000D-02
-6.4800000D-02  -4.3600000D-01  -7.6000000D-01   0.0000000D+00
 0.0000000D+00   0.0000000D+00   1.0000000D+00   0.0000000D+00
```

```
EXAMPLE TIMFST RUN
 ORACLS  PROGRAM

    B     MATRIX       4 ROWS       2 COLUMNS
 0.0000000D+00   9.6000000D-03
 0.0000000D+00  -1.1300000D-01
-1.5000000D+00  -5.5000000D-02
 0.0000000D+00   0.0000000D+00
```

Figure D.1-10: Example TIMFST Run - Output File

```
EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


   XI    MATRIX       8 ROWS       1 COLUMNS
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00


EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    D     MATRIX       4 ROWS       2 COLUMNS
 0.0000000D+00   0.0000000D+00
 0.0000000D+00   0.0000000D+00
 0.0000000D+00   1.6200000D-02
 0.0000000D+00  -1.9000000D-01


EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


   C2    MATRIX       2 ROWS       4 COLUMNS
 0.0000000D+00   0.0000000D+00   0.0000000D+00   1.0000000D+00
-9.8500000D-02  -1.6850000D+00   0.0000000D+00   2.3500000D+00
```

Figure D.1-10: Example TIMFST Run - Output File (Continued)

```
EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    C     MATRIX       4 ROWS       4 COLUMNS
 1.0000000D+00   0.0000000D+00   0.0000000D+00   0.0000000D+00
 0.0000000D+00   0.0000000D+00   0.0000000D+00   1.0000000D+00
-1.0800000D-02   1.9900000D-01   0.0000000D+00   0.0000000D+00
-5.2300000D-01  -7.6800000D-01  -4.5400000D-03   0.0000000D+00

EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    J     MATRIX       2 ROWS       4 COLUMNS
 1.0000000D-01   1.0000000D+00   0.0000000D+00   0.0000000D+00
-1.0000000D-02   0.0000000D+00   0.0000000D+00   0.0000000D+00


EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    E     MATRIX       4 ROWS       4 COLUMNS
-1.3280000D-01   1.1800000D-01  -5.4400000D-02   2.7900000D-02
-2.6890000D-01  -4.5500000D-01   1.4000000D+00   8.0000000D-04
 7.2300000D-02  -4.3600000D-01  -7.6000000D-01  -3.6000000D-03
 5.3600000D-02   0.0000000D+00   1.0000000D+00  -6.5000000D-02


EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    F     MATRIX       2 ROWS       4 COLUMNS
 0.0000000D+00   0.0000000D+00   1.0000000D-01   0.0000000D+00
 0.0000000D+00   0.0000000D+00   0.0000000D+00   0.0000000D+00
```

Figure D.1-10: Example TIMFST Run - Output File (Continued)

```
EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    G     MATRIX       4 ROWS       4 COLUMNS
-2.5100000D-02  -4.6500000D-01   3.9000000D-03  -1.9600000D-02
-6.5070000D-02   2.8400000D+00  -2.7900000D-02   3.6300000D-01
-1.0330000D-01  -1.0850000D+00  -1.9500000D-02   1.6700000D-01
-1.7200000D-02   1.7860000D+00  -7.8000000D-03   8.3500000D-02


EXAMPLE TIMFST RUN
 ORACLS  PROGRAM


    H     MATRIX       4 ROWS       2 COLUMNS
 1.0000000D+00   0.0000000D+00
 0.0000000D+00   0.0000000D+00
 0.0000000D+00   0.0000000D+00
 0.0000000D+00   1.0000000D+00


*****************************************************
******************** BEGIN DEBUG ********************
*****************************************************
 THE FOLLOWING INFORMATION IS FOR DEBUG PURPOSES.
 SEE IAC MANUAL SECTION D.1.2 FOR EXPLANATION
 OF FOLLOWING MATRICES.
```

Figure D.1-10: Example TIMFST Run - Output File (Continued)

...

****
********** END DEBUG **********
****

EIGENVALUES OF CLOSED LOOP DYNAMICS MATRIX

EIGENVALUES			
REAL	IMAGINARY	DAMP RATIO	FREQUENCY
-0.01990422	0.00000000	1.00000000	0.01990422
-0.17492237	0.00000000	1.00000000	0.17492237
-0.14636817	0.11317448	0.79109720	0.18501919
-0.14636817	-0.11317448	0.79109720	0.18501919
-0.55088873	0.73164082	0.60150704	0.91584752
-0.55088873	-0.73164082	0.60150704	0.91584752
-0.52247781	1.43028474	0.34311982	1.52272700
-0.52247781	-1.43028474	0.34311982	1.52272700

Plant states are STATE 1 thru STATE 4

Estimated states are STATE 5 thru STATE 8

Plant outputs are OUTPUT 1 thru OUTPUT 4

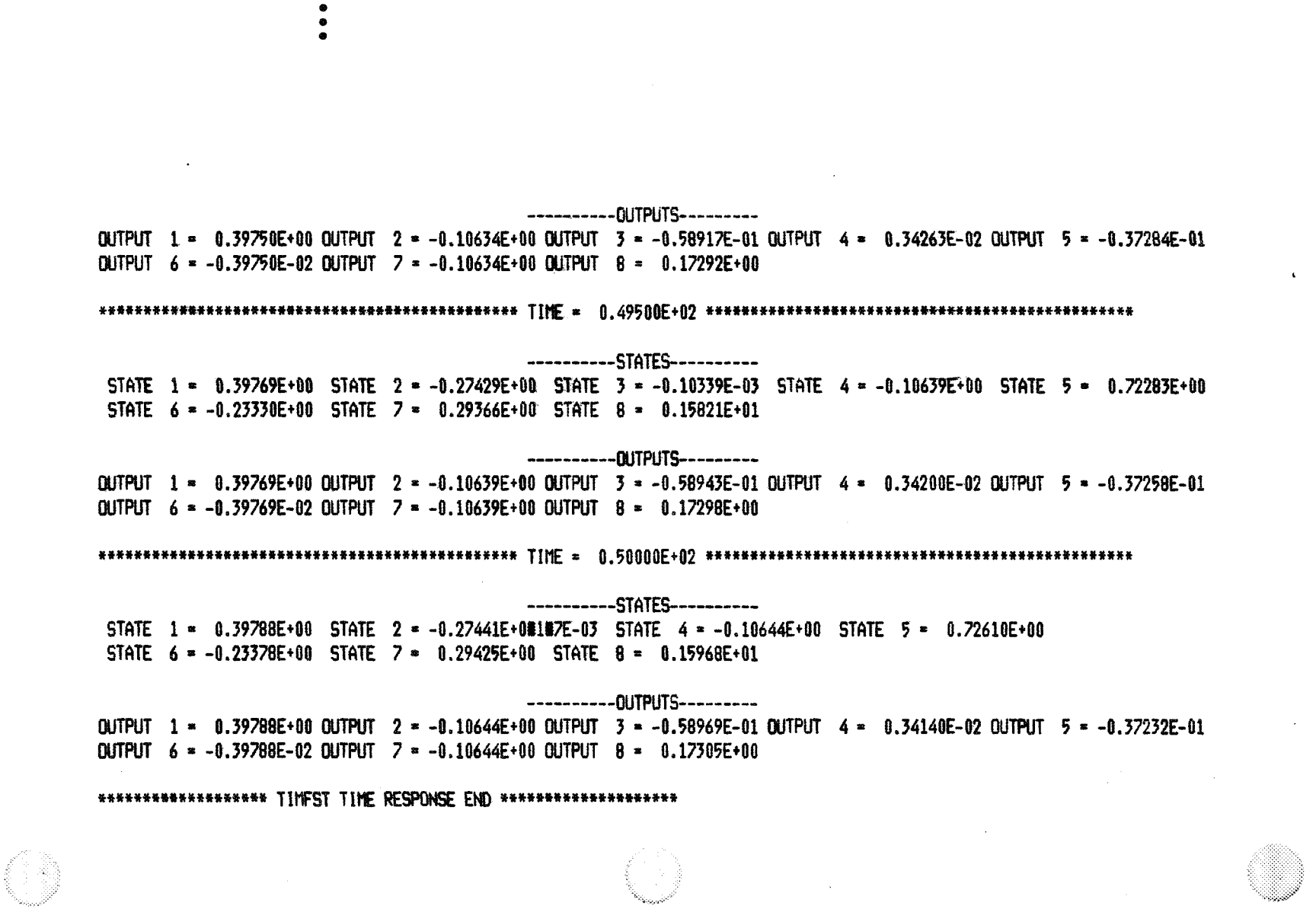
Controller outputs are OUTPUT 5 thru OUTPUT 6

Other outputs are OUTPUT 7 thru OUTPUT 8

Figure D.1-10: Example TIMFST Run - Output File (Continued)

```
EXAMPLE TIMFST RUN                                                    ORACLS  PROGRAM


    U     MATRIX       6 ROWS       1 COLUMNS
 1.0000000D-01
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00
 0.0000000D+00

                                    TIME RESPONSE

********************************************** TIME =  0.00000E+00 **********************************************

                                         ---------STATES---------
 STATE  1 =  0.00000E+00  STATE  2 =  0.00000E+00  STATE  3 =  0.00000E+00  STATE  4 =  0.00000E+00  STATE  5 =  0.00000E+00
 STATE  6 =  0.00000E+00  STATE  7 =  0.00000E+00  STATE  8 =  0.00000E+00

                                         ---------OUTPUTS---------
OUTPUT  1 =  0.00000E+00 OUTPUT  2 =  0.00000E+00 OUTPUT  3 =  0.00000E+00 OUTPUT  4 =  0.00000E+00 OUTPUT  5 =  0.00000E+00
OUTPUT  6 =  0.00000E+00 OUTPUT  7 =  0.00000E+00 OUTPUT  8 =  0.00000E+00

********************************************** TIME =  0.50000E+00 **********************************************

                                         ---------STATES---------
 STATE  1 =  0.13516E-02  STATE  2 = -0.20714E-01  STATE  3 = -0.57268E-01  STATE  4 = -0.15909E-01  STATE  5 =  0.46741E-01
 STATE  6 = -0.86784E-02  STATE  7 =  0.44363E-02  STATE  8 = -0.34516E-02

                                         ---------OUTPUTS---------
OUTPUT  1 =  0.13516E-02 OUTPUT  2 = -0.15909E-01 OUTPUT  3 = -0.41370E-02 OUTPUT  4 =  0.15464E-01 OUTPUT  5 = -0.15330E-01
OUTPUT  6 = -0.13516E-04 OUTPUT  7 = -0.15909E-01 OUTPUT  8 = -0.26160E-02

********************************************** TIME =  0.10000E+01 **********************************************

                                         ---------STATES---------
 STATE  1 =  0.62241E-02  STATE  2 = -0.60701E-01  STATE  3 = -0.74283E-01  STATE  4 = -0.50429E-01  STATE  5 =  0.82275E-01
 STATE  6 = -0.43178E-01  STATE  7 =  0.26752E-01  STATE  8 = -0.22198E-01
```

Figure D.1-10: Example TIMFST Run - Output File (Continued)



```
                                                ----------OUTPUTS---------
OUTPUT  1 =  0.39750E+00 OUTPUT  2 = -0.10634E+00 OUTPUT  3 = -0.58917E-01 OUTPUT  4 =  0.34263E-02 OUTPUT  5 = -0.37284E-01
OUTPUT  6 = -0.39750E-02 OUTPUT  7 = -0.10634E+00 OUTPUT  8 =  0.17292E+00

********************************************** TIME =  0.49500E+02 **********************************************

                                                ----------STATES----------
 STATE  1 =  0.39769E+00  STATE  2 = -0.27429E+00  STATE  3 = -0.10339E-03  STATE  4 = -0.10639E+00  STATE  5 =  0.72283E+00
 STATE  6 = -0.23330E+00  STATE  7 =  0.29366E+00  STATE  8 =  0.15821E+01

                                                ----------OUTPUTS---------
OUTPUT  1 =  0.39769E+00 OUTPUT  2 = -0.10639E+00 OUTPUT  3 = -0.58943E-01 OUTPUT  4 =  0.34200E-02 OUTPUT  5 = -0.37258E-01
OUTPUT  6 = -0.39769E-02 OUTPUT  7 = -0.10639E+00 OUTPUT  8 =  0.17298E+00

********************************************** TIME =  0.50000E+02 **********************************************

                                                ----------STATES----------
 STATE  1 =  0.39788E+00  STATE  2 = -0.27441E+00  STATE  3 = -0.10407E-03  STATE  4 = -0.10644E+00  STATE  5 =  0.72610E+00
 STATE  6 = -0.23378E+00  STATE  7 =  0.29425E+00  STATE  8 =  0.15968E+01

                                                ----------OUTPUTS---------
OUTPUT  1 =  0.39788E+00 OUTPUT  2 = -0.10644E+00 OUTPUT  3 = -0.58969E-01 OUTPUT  4 =  0.34140E-02 OUTPUT  5 = -0.37232E-01
OUTPUT  6 = -0.39788E-02 OUTPUT  7 = -0.10644E+00 OUTPUT  8 =  0.17305E+00

******************** TIMFST TIME RESPONSE END ********************
```

Figure D.1-10: Example TIMFST Run - Output File (Continued)

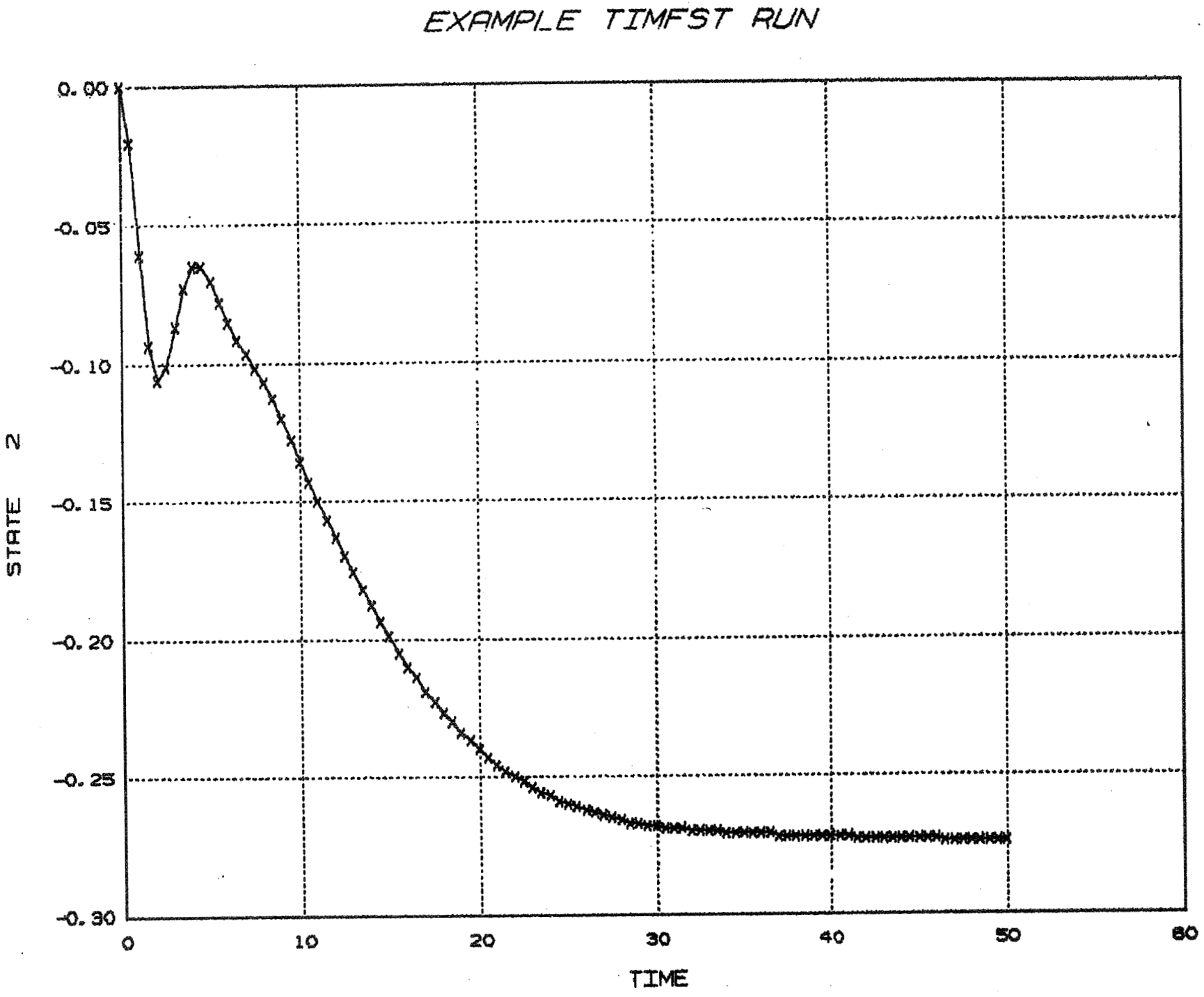


Figure D.1-11: Example TIMFST Run - Plot

D.2 USER-DEFINED DRIVERS

The ORACLS program can be executed within IAC employing a user-defined driver and a user-defined dataset. Matrix data can be read by the ORACLS program from the database using the ORAGET call and matrix data may be stored in the database using the ORAPUT call. In addition data can be read and printed for output using the standard ORACLS input/output routines.

The RUN statement for a user-defined ORACLS program may be of the form

```
RUN ORACLS (EXE = filename, F = filename, FIN = filespec,+
PRT = option)
```

The EXE parameter names the executable file (filename.EXE) output from the linker. The F parameter names the input file (filename.DAT) and the output file (filename.LIS). FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec filename.DAT. The optional PRT parameter specifies disposition of the output file.

Utility Routines - In building an ORACLS driver program for execution within IAC, there are several subroutine calls inserted into the driver. The subroutine call sequences and explanations are discussed in Section D.3. Care must be taken to allocate or dimension matrices large enough in the driver program to avoid erroneous results. Consult the ORACLS manual for details on the necessary dimensions, variables, and dummy arrays.

Linking Driver - The IAC command to link a user-defined driver program with the standard ORACLS software is

```
LINK (MOD = ORACLS, EXE = exec-file, OBJ = obj-files, LIB = lib-files)
```

where exec-file is the name of the LINK output executable load-module file (type EXE) to be used in a later RUN ORACLS command, obj-files is a list of the compiled main program and other user routines (type OBJ) supplied by the user, and lib-files is a list of object libraries (type OLB) supplied by the user.

D.3 NEW IAC ORACLS UTILITY PROCEDURES

Outlined below are IAC provided utility function and subroutine procedures which support both standard and user-defined ORACLS drivers. Procedures beginning with the characters IAC are IAC standard utilities, and those beginning with ORA are IAC provided utilities specific to ORACLS. Procedures marked with an asterisk (*) are required by all drivers, and the others are optional.

Utility Procedure	Explanation
* IACBEG ('ORACLS',QUAL, FUNITS,WUNITS,NSIZE,CSIZE)	An ORACLS driver program must begin with the IACBEG call to initialize communication with the IAC utilities. See Section 6 for description of the calling arguments.
ORAFIL(FI,FO,AREA)	ORAFIL retrieves the ORACLS input and output filespecs, and the array storage area code, defined via parameters in the IAC RUN command. The input-file and output-file values are character strings dimensioned sufficiently long in the ORACLS driver program to hold the file specs. The output from ORACLS goes to file unit 6. Therefore the output-file should be opened to unit 6 prior to calling any ORACLS routines which may cause output. The input-file must always exist. The area code is one of the character values D, H, DH or HD.
RDTITL	RDTITL reads from the terminal an 80-character string to use for the page heading, and also initializes several constants. It should always be the next

	ORACLS subroutine called after IACBEG and ORAFIL.
ORADEF(NAMTAG,LOC,NEL)	Sets up storage for an array in the IAC ORACLS workspace. NAMTAG is a maximum 10 character id defining a name for the array. LOC is the returned location of the array within the IAC numeric workspace, and NEL is the given number of double precision storage locations needed for the array.
ORADEL(NAMTAG)	Deletes an array from the IAC ORACLS workspace. NAMTAG is the array id (e.g. assigned via a call to ORADEF or ORAGET).
ORAGET(AREA,SPEC,NAMTAG,LOC, SIZE)	Gets an ORACLS array, from the IAC database or a host directory. AREA is the given data storage area code, SPEC is the array spec, and NAMTAG is the array id. LOC is the returned location of the array within the IAC numeric workspace, and SIZE(2) is a returned vector containing the sizes of array index 1 and 2.
ORAPAC(AP,NAP,A,KNR,NR,NC)	Converts on unpacked matrix A into packed form AP.
ORAPUT(AREA,SPEC,NAMTAG,LOC, SIZE)	Puts an ORACLS array, to the IAC database or a host directory. AREA is the given data storage area code, SPEC is the array spec, NAMTAG is the array id, LOC is the location of the array within the IAC numeric workspace, and SIZE(2) is a given vector containing the sizes of array index 1 and 2.

ORAQRE(UIN,LF,HOLD)	Performs a VAX host Q-read operation. UIN is the given input file unit number, LF is the returned length of the character string read, and HOLD is the returned character string.
ORAQUE(PROMPT)	A logical function which prints the given character string prompt on the terminal, and accepts a YES or NO answer from the user. The returned function is true for YES, false for NO.
ORATIA(T,NT,A,NA,TIA,NTIA)	Performs the matrix operation TIA = T^{-1} * A. The input matrix T is destroyed upon return.
ORAUNP(A,KNR,NR,NC,AP,NAP)	Converts a packed matrix AP into unpacked form A.
ORALIS(UOUT,HEAD)	Displays a list of all arrays currently defined in the IAC ORACLS workspace (may be used for debussing purposes). UOUT is the given output unit number for display, and HEAD is a given character string heading to be printed at the top of the display.
* IACEND(QUAL)	An ORACLS driver program must end with the IACEND call to terminate communication with the IAC utilities.

D.4 MODIFICATION AND OPERATION OF ORIGINAL ORACLS PACKAGE

ORACLS Routines - In the IAC ORACLS implementation, two subroutines have been added to the original list documented in References 13 and 14. Subroutine CDIV was added to perform double precision complex division for HQR2 and INVIT. Subroutine ELTRAN is discussed in Reference 12 and provides additional eigenanalysis solution paths. The current list of IAC ORACLS subroutines is given below.

IAC ORACLS Subroutine List

ADD	ASYFIL	ASYREG	ATXPXA	AXPXB
BALANC	BALBAK	BARSTW	BCKMLT	BILIN
CDIV	CNTREG	CSTAB	CTROL	DETFAC
DISREG	DSTAB	EIGEN	ELMBAK	ELMHES
ELTRAN	EQUATE	EXMDFL	EXPADE	EXPINT
EXPSER	FACTOR	GAUSEL	GELIM	HQR
HQR2	HSHLDR	IMMDFL	INVIT	JUXTC
JUXTR	LEVIER	LNCNT	MAXEL	MULT
NORMS	NULL	POLE	PREFIL	PRNT
RDTITL	READ	READ1	RICNWT	SAMPL
SCALE	SCHUR	SHRSLV	SNVDEC	SUBT
SUM	SYMPDS	SYMSLV	SYSSLV	TESTST
TRANP	TRCE	TRNSIT	UNITY	VARANC

Calls to LINPACK Routines - The ORACLS library of subroutines is self contained with one exception. Calls to LINPACK subroutines have been inserted into certain ORACLS subroutines to perform some of the linear algebra. Therefore to run a user-defined ORACLS driver program, it must first be linked with the LINPACK library. The LINPACK Users Guide (Reference 15) describes all LINPACK routines including complete subroutine listings. The LINPACK routines used by ORACLS are summarized below.

Primary LINPACK Subroutines Called by ORACLS

DSVDC	DGECO	DGESL	DPOCO	DPOSL
DSICO				

Basic Linear Algebra Subroutines Called by Above Subroutines

DDOT	DAXPY	DROTG	DROT	DCOPY
DSWAP	DNRM2	SASUM	DSCAL	DMAX

Special Modifications - The ORACLS subroutines also require a function subroutine called DAMCON which defines machine epsilon and is thus machine dependent. The function DAMCON should be defined as DAMCON(3) = machine epsilon. The other elements of DAMCON are not required.

The ORACLS matrix print routine has been modified to output matrices by columns with a fixed set of 7 columns per line.

Subroutine Name Changes - Original names of the ORACLS subroutines defined in References 13 and 14 have been reduced to a maximum of 6 characters. This name change affects the following routines.

Original Name	New Name
TESTSTA	TESTST
VARANCE	VARANC
TRANSIT	TRNSIT
DISCREG	DISREG
CNTNREG	CNTREG
RICTNWT	RICNWT
ASYMREG	ASYREG
ASYMFIL	ASYFIL
EXPMDFL	EXMDFL
IMPMDFL	IMMDFL

APPENDIX E - MIMIC IMPLEMENTATION

This appendix provides theory and example problems for the MIMIC (Model Integration via Mesh Interpolation Coefficients) module. The MIMIC RUN statement and data flow are described in Section 3.6. MIMIC transforms field values (e.g. temperatures, pressures or stress tensors) from one model to another. The transformation involves coordinate based interpolation over 1-D, 2-D, or 3-D metric space. The naming convention used by MIMIC is that mesh A is associated with existing field values, and mesh B is associated with field values to be computed.

E.1 INTERPOLATION THEORY

The interpolation is performed by MIMIC using a linear interpolation algorithm over 1-D, 2-D and/or 3-D simplex regions. This involves respective lines, triangles and tetrahedra, as appropriate for the particular mesh points involved. For example, the 2-D interpolation is used for 2-dimensional regions, or within certain portions of 3-dimensional regions where special conditions exist. Arbitrary coordinates can be handled, for example: time TIME; distance X; 2-D mesh cordinates X, Y; 3-D mesh coordinates X, Y, Z; etc. MIMIC relies only upon interpolation, and does not perform extrapolation.

1-D Interpolation - The 1-dimensional line interpolation is rather trivial. For each B-point it involves the selection of two A-points, one on either side. The corresponding two interpolation coefficients are computed via the relative distance ratios. If two such A-points can not be found, a single point is selected and the corresponding interpolation coefficient is set to the value 1.0.

2-D Interpolation - The 2-dimensional interpolation scheme is illustrated by Figure E.1-1. There the open circles represent points from the A-mesh, and the closed circle (point N) represents one of the points from the B-mesh. In general, three A-points must be selected (forming an enclosing triangle around the B-point) in order to accomplish an exact linear interpolation. The sequence for selecting these three points is as follows.

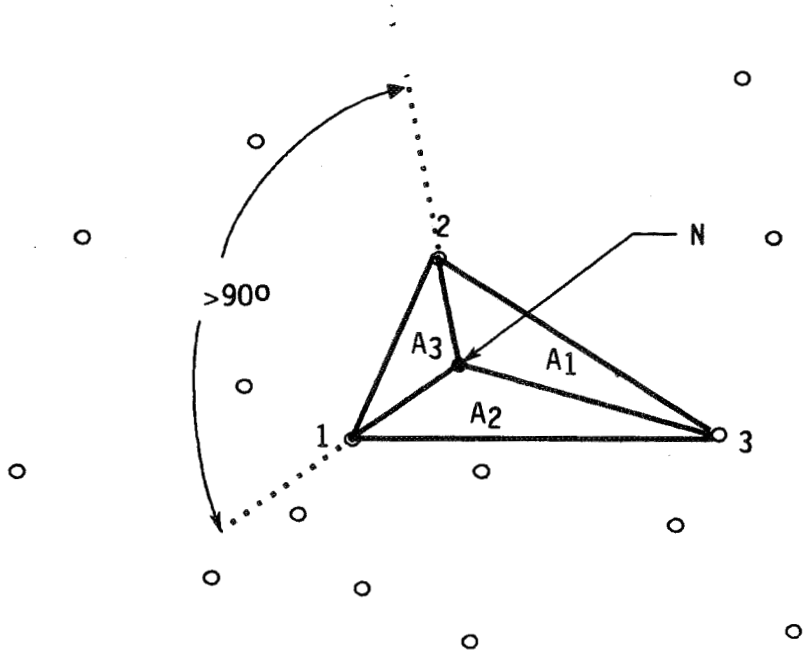


Figure E.1-1: MIMIC 2-D Spatial Interpolation

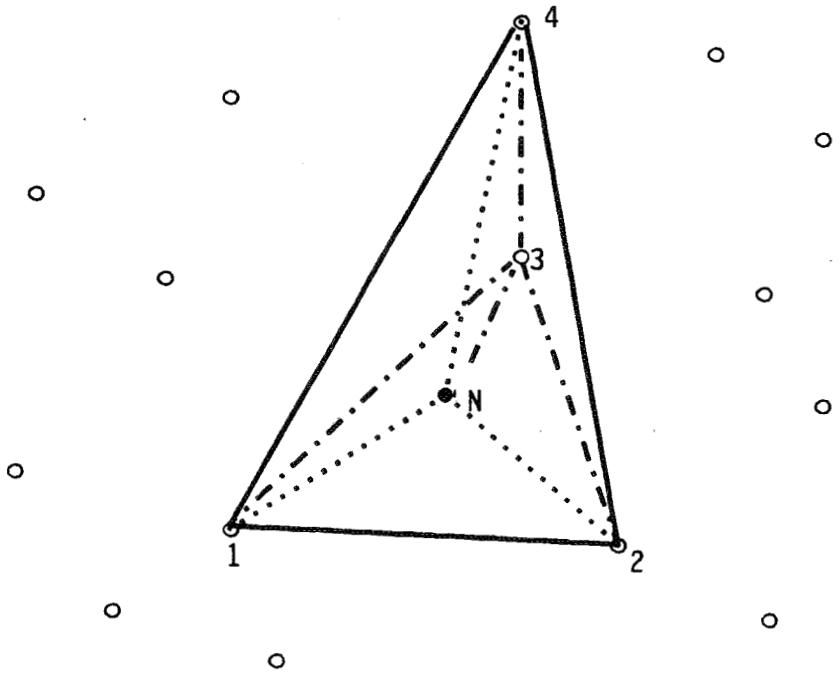


Figure E.1-2: MIMIC 3-D Spatial Interpolation

1. Find point 1 as the A-point closest to N.
2. Locate point 2 as the next closest A-point such that the smaller angle between rays N-1 and N-2 is greater than 90°. (The restriction on the angle is necessary to avoid possible selection later of the third point such that a long narrow triangle would be formed. Note it also ensures that a perpendicular from N to the line 1-2 will pass between points 1 and 2).
3. Locate point 3 as the next closest A-point such that N is enclosed within the triangle 1-2-3. (A necessary and sufficient condition for satisfying this requirement is that the three vector cross products N-1 X N-2, N-2 X N-3, and N-3 X N-1, all have the same direction).

The B-point N divides the triangle into three triangular sub-areas A1, A2, and A3, with total area of the triangle defined by $A = A_1 + A_2 + A_3$. Interpolation coefficients (weights) are then defined by $W_1 = A_1/A$, $W_2 = A_2/A$, and $W_3 = A_3/A$, so that $W_1 + W_2 + W_3 = 1$. These coefficients allow the value of a field variable, T_N, at point N to be computed as

$$T_N = W_1T_1 + W_2T_2 + W_3T_3 \qquad \text{(E.1-1)}$$

where T_i is the field value at the ith vertex point of the triangle.

In case a third A-point is not found to form an enclosing traingle, a perpendicular is defined from N to the line 1-2, intersecting 1-2 at say point N^1. The coefficients W_1 and W_2 are then computed as the distances from N^1 to points 2 and 1, respectively, divided by the length of line 1-2. T_N can still be computed from Equation E.1-1 (using $W_3 = 0$, with point 3 = point 1). In case a second point is also not found according to the above selection procedure, T_N can again be computed from Equation E-1 (using $W_2 = W_3 = 0$, with point 2 = point 3 = point 1).

<u>3-D Interpolation</u> - The 3-dimensional interpolation scheme is similar to the 2-dimensional scheme, except that tetrahedra rather than triangles are involved. As shown by Figure E.1-2, the open circles again represent points from the A-mesh, and the closed circle (point N) represents one of the points from the B-mesh. Four A-points must now be selected to form an enclosing

tetrahedron around the B-point in order to accomplish an exact linear interpolation. The sequence for selecting these four points is as follows.

1. Find point 1 as the A-point closest to N.
2. Locate point 2 as the next closest A-point such that the smaller angle between lines N-1 and N-2 is greater than 90°.
3. Locate point 3 as the next closest A-point such that a perpendicular from N to the plane 1-2-3 intersects this plane within the triangle 1-2-3. (A vector S is defined normal to the plane 1-2-3. The requirement is satisfied if the three vector dot-cross products $S \cdot (N\text{-}1 \times N\text{-}2)$, $S \cdot (N\text{-}2 \times N\text{-}3)$, and $S \cdot (N\text{-}3 \times N\text{-}1)$, all have the same sign.)
4. Locate point 4 as the next closest A-point such that N is enclosed within the tetrahedron 1-2-3. (A necessary and sufficient condition for satisfying this requirement is that the four vector dot-cross products $-\ N\text{-}1 \cdot (N\text{-}3 \times N\text{-}4)$, $+\ N\text{-}2 \cdot (N\text{-}4 \times N\text{-}1)$, $-\ N\text{-}3 \cdot (N\text{-}1 \times N\text{-}2)$, and $+\ N\text{-}4 \cdot (N\text{-}2 \times N\text{-}3)$, all have the same sign.)

The B-point N divides the tetrahedron into four tetrahedral sub-volumes, V_1, V_2, V_3, and V_4, with total volume of the tetrahedron defined by $V = V_1 + V_2 + V_3 + V_4$ (V_i is the volume of the tetrahedron opposite vertex point i). Interpolation coefficients are then defined by $W_1 = V_1/V$, $W_2 = V_2/V$, $W_3 = V_3/V$, and $W_4 = V_4/V$, so that $W_1 + W_2 + W_3 + W_4 = 1$. These coefficients allow the value of a field variable, T_N, at point N to be computed as

$$T_N = W_1T_1 + W_2T_2 + W_3T_3 + W_4T_4 \qquad \text{(E.1-2)}$$

where T_i is the field value at the ith vertex point of the tetrahedron.

In case a fourth A-point is not found to form an enclosing tetrahedron, point N is projected perpendicularly onto the 1-2-3 plane, and the 2-dimensional interpolation scheme using triangle 1-2-3 is employed. In case a third or second point is also not found, MIMIC logic again proceeds as discussed for 2-dimensional interpolation.

Limitations - The MIMIC interpolation scheme tends to be robust in nature, and it produces acceptable results in most situations. However, because the interpolation algorithm is based on closeness of points and because no extrapolation is used, the analyst should be aware that the following situations can sometimes produce inaccurate results.

1) Mesh B contains points which are significantly outside the boundaries of mesh A.
2) Mesh A does not have enough points to adequately describe the field gradients.
3) Sharp discontinuities exist in the field.
4) Coordinates have different metric units.
5) Field gradients are much greater in some directions than in others, and/or points are much more closely spaced in some directions than in others.

Computation Process - The field value interpolation computation defined by Equations E.1-1 and/or E.1-2 can be summarized, in matrix form, for all points by

B-values = C-coefficients times A-values (E.1-3)

The A-values and B-values are stored in arrays; the values are of type real single or double precision, either scalars or nonscalars. The C-coefficients are stored in a relation; the coefficients consist of the real double precision coefficients W_i described above, and associated integer values which reference positions within an index of the A-array. In transforming the A-array to the B-array, the referenced index of the A-array is replaced by the index of the C-relation. More than one interpolation attribute (e.g. both temperature and strain tensor) may be contained in the A-array and B-array, in which case MIMIC uses the same C-relation to interpolate all attributes simultaneously.

E.2 EXAMPLE PROBLEMS

Two examples, a 2-dimensional problem and a 3-dimensional problem, are presented in this section. Since these examples were intended to serve as

checkout cases and to illustrate MIMIC operation, the models were specially defined to facilitate user visualization and manual checking.

2-D Problem - The X-Y spatial models used in this problem are shown overlayed in Figure E.2-1. The open circles represent points in the A-mesh, and closed circles represent points in the B-mesh. The respective A and B X-Y mesh definitions are listed in Figure E.2-2 a and b.

Table E.2-1 a and b defines field values for the A and B mesh points, respectively, which can be used in a convenient test case to exercise the computed coefficients relation. Field values, T, in this table are computable as a function of position coordinates:

$$T = 50X - 100Y$$

3-D Problem - The X-Y-Z spatial models for this problem can be described relative to three parallel planes, stacked on top of one another in three levels, as shown in Figure E.2-3. The arrangement of the points on each respective plane is shown in Figure E.2-4. There the open circles represent points in the A-mesh, and closed circles represent points in the B-mesh. The respective A and B X-Y-Z mesh definitions are listed in Figure E.2-5 a and b. As indicated by the listed Z-coordinate values, the B-points are at levels 1.0, 2.0 and 3.0; the A-points are offset slightly, at levels .999, 1.999 and 3.001.

Table E.2-2 a and b defines field values for the respective A and B mesh points, which can be used in a test case to exercise the computed coefficients relation. Exact field values, T, in this table are computable as a function of position coordinates:

$$T = 50X - 100Y + 50Z$$

Note that MIMIC computed field values differ from exact values where the B-point could not be enclosed within a tetrahedron formed by four A-points. This is true for B-points 12, 13, 16, 20, 26, 30, 31, 32, 33, 35 and 38.

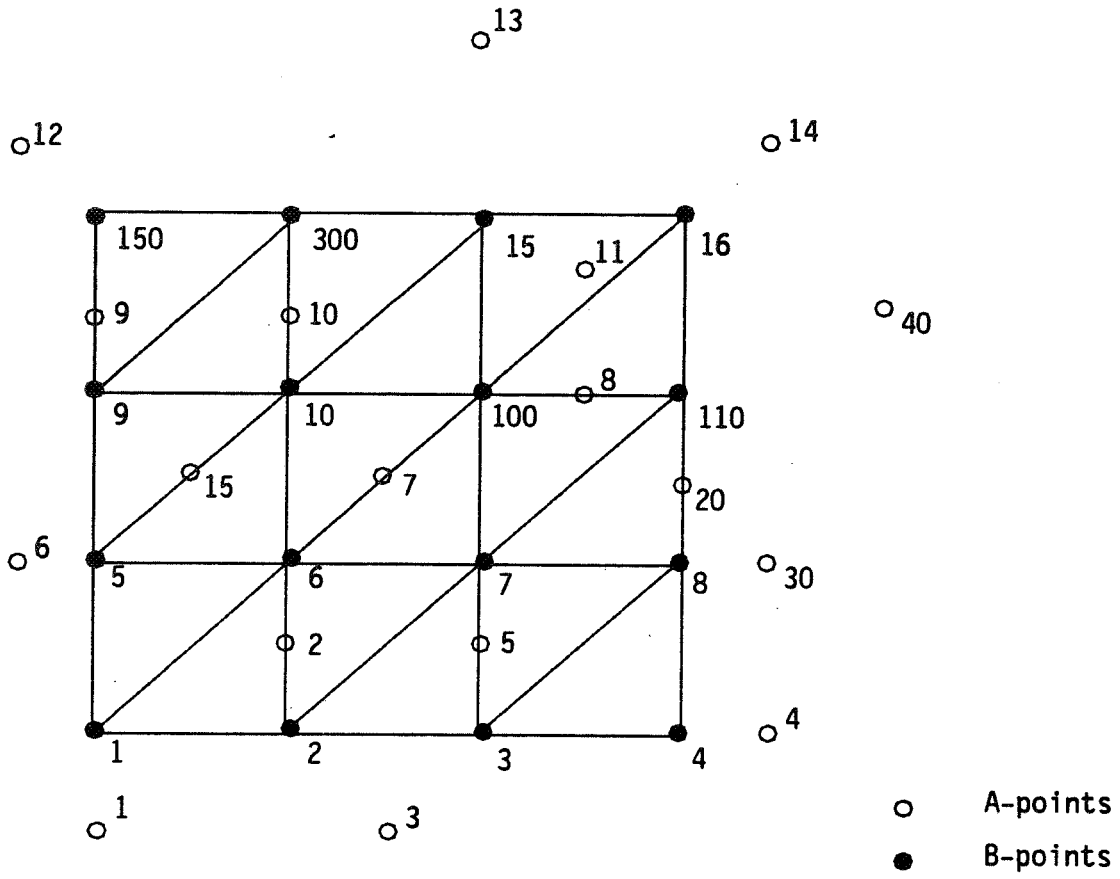


Figure E.2-1: Meshes for MIMIC 2-D Example

Point	X	Y
1	0.000	-0.500
2	1.000	0.500
3	1.500	-0.500
4	3.500	0.000
5	2.000	0.500
6	-0.500	1.000
7	1.500	1.500
30	3.500	1.000
15	0.500	1.500
20	3.000	1.500
40	4.000	2.500
12	-0.500	3.500
8	2.500	2.000
9	0.000	2.500
10	1.000	2.500
11	2.500	2.750
13	2.000	4.000
14	3.500	3.500

(a) A-mesh coordinates

Point	X	Y
1	0.000	0.000
2	1.000	0.000
3	2.000	0.000
4	3.000	0.000
5	0.000	1.000
6	1.000	1.000
7	2.000	1.000
8	3.000	1.000
9	0.000	2.000
10	1.000	2.000
100	2.000	2.000
110	3.000	2.000
150	0.000	3.000
300	1.000	3.000
15	2.000	3.000
16	3.000	3.000

(b) B-mesh Coordinates

Figure E.2-2: Mesh Coordinates for MIMIC 2-D Example

Point	Value
1	50.
2	0.
3	125.
4	175.
5	50.
6	-125.
7	-75.
30	75.
15	-125.
20	0.
40	-50.
12	-375.
8	-75.
9	-250.
10	-200.
11	-150.
13	-300.
14	-175.

(a) A-Field Values

Point	Value
1	0.
2	50.
3	100.
4	150.
5	-100.
6	-50.
7	0.
8	50.
9	-200.
10	-150.
100	-100
110	-50.
150	-300.
300	-250.
15	-200
16	-150.

(b) B-Field Values

Table E.2-1: Field Values for MIMIC 2-D Example

Point	Value
1	24.95
2	25.05
3	75.05
4	0.00
5	62.45
6	-124.95
7	-75.05
8	-175.15
9	-75.15
10	150.00
11	174.95
12	175.05
13	-0.10
14	24.95
15	124.95
16	-100.10
17	75.05
18	-75.15
19	-25.15
20	-150.05
21	175.15
22	75.05
23	25.05
24	-50.00
25	50.10
26	-174.95
27	-124.95

(a) A-Field Values

Point	Exact Value	Mimic Value
10	0.00	0.00
11	50.00	50.00
12	100.00	80.60
13	-100.00	-110.02
14	-50.00	-50.00
15	0.00	0.00
16	-200.00	-162.54
17	-150.00	-150.00
18	-100.00	-100.00
20	50.00	35.24
21	100.00	100.00
22	150.00	150.00
23	-50.00	-50.00
24	0.00	0.00
25	50.00	50.00
26	-150.00	-112.55
27	-100.00	-100.00
28	-50.00	-50.00
30	100.00	66.69
31	150.00	134.56
32	200.00	161.29
33	0.00	-16.64
34	50.00	50.00
35	100.00	45.52
36	-100.00	-100.00
37	-50.00	-50.00
38	0.00	29.97

(b) B-Field Values

Table E.2-2: Field Values for MIMIC 3-D Example

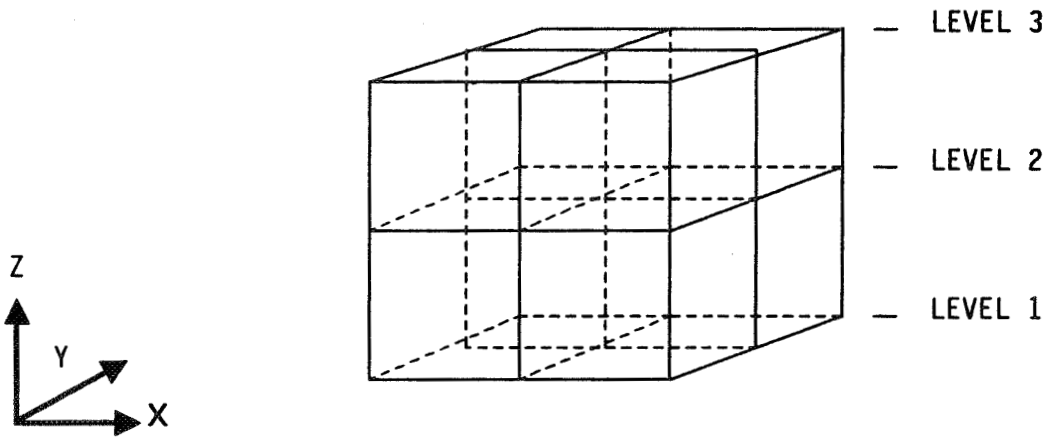


Figure E.2-3: Geometry for MIMIC 3-D Example

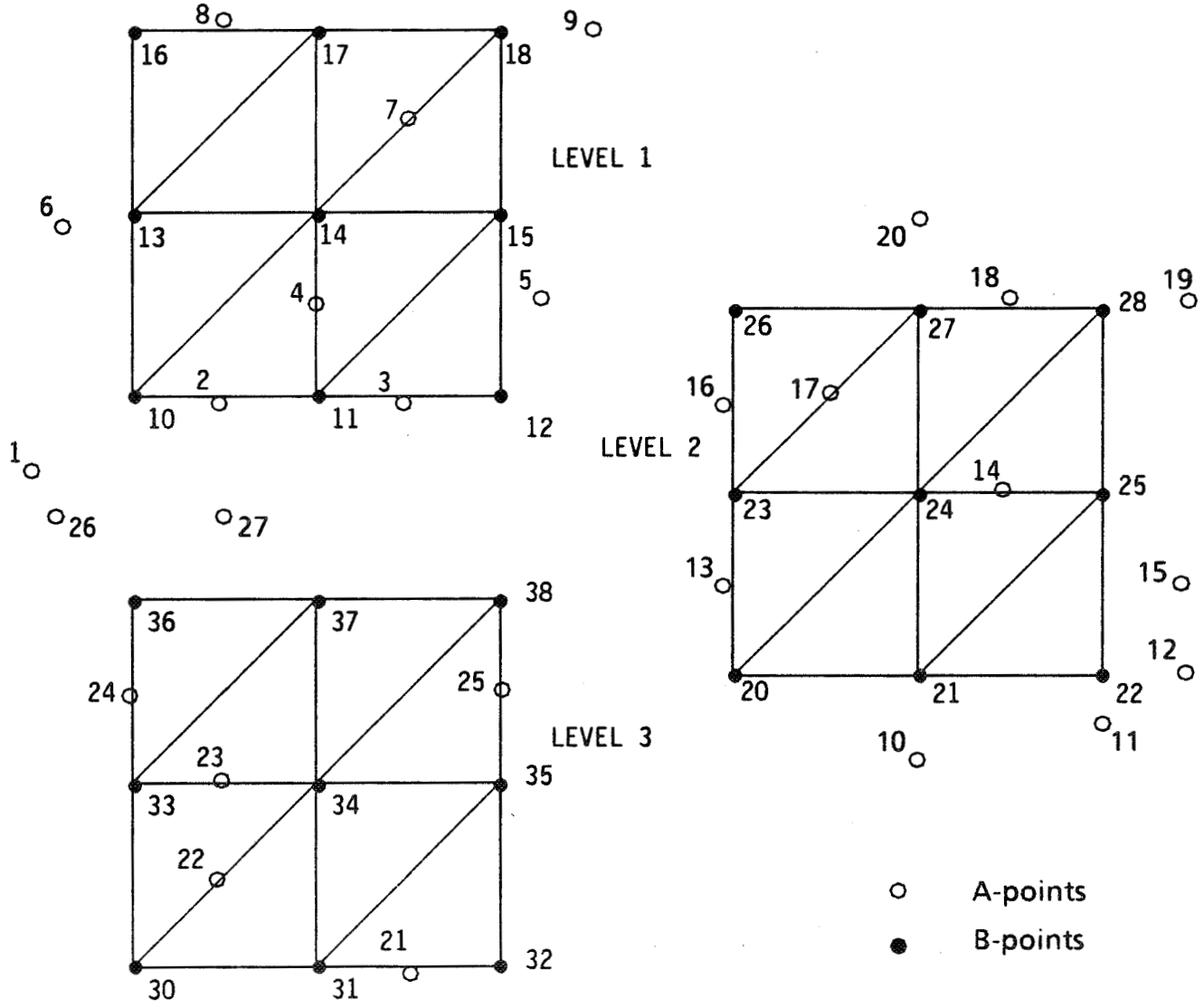


Figure E.2-4: Mesh Points for MIMIC 3-D Example

Point	X	Y	Z
1	0.500	0.500	0.999
2	1.500	0.999	0.999
3	2.500	0.999	0.999
4	2.001	1.500	0.999
5	3.250	1.500	0.999
6	0.500	1.999	0.999
7	2.500	2.500	0.999
8	1.500	3.001	0.999
9	3.500	3.001	0.999
10	2.001	0.500	1.999
11	3.000	0.750	1.999
12	3.500	0.999	1.999
13	0.999	1.500	1.999
14	2.500	2.000	1.999
15	3.500	1.500	1.999
16	0.999	2.500	1.999
17	1.500	2.500	1.999
18	2.500	3.001	1.999
19	3.500	3.001	1.999
20	2.000	3.500	1.999
21	2.500	0.999	3.001
22	1.500	1.500	3.001
23	1.500	2.000	3.001
24	0.999	2.500	3.001
25	3.001	2.500	3.001
26	0.500	3.500	3.001
27	1.500	3.500	3.001

(a) A-Mesh Coordinates

Point	X	Y	Z
10	1.000	1.000	1.000
11	2.000	1.000	1.000
12	3.000	1.000	1.000
13	1.000	2.000	1.000
14	2.000	2.000	1.000
15	3.000	2.000	1.000
16	1.000	3.000	1.000
17	2.000	3.000	1.000
18	3.000	3.000	1.000
20	1.000	1.000	2.000
21	2.000	1.000	2.000
22	3.000	1.000	2.000
23	1.000	2.000	2.000
24	2.000	2.000	2.000
25	3.000	2.000	2.000
26	1.000	3.000	2.000
27	2.000	3.000	2.000
28	3.000	3.000	2.000
30	1.000	1.000	3.000
31	2.000	1.000	3.000
32	3.000	1.000	3.000
33	1.000	2.000	3.000
34	2.000	2.000	3.000
35	3.000	2.000	3.000
36	1.000	3.000	3.000
37	2.000	3.000	3.000
38	3.000	3.000	3.000

(b) B-Mesh Coordinates

Figure E.2-5: Mesh Coordinates for MIMIC 3-D Example

APPENDIX F - IAC SOLUTION PATH IV METHODOLOGY

F.1 THEORY

The IAC Solution Path IV (see also Section 4.4) provides for inclusion of time varying quasi-static thermal loading effects, within a coupled structural/control time domain analysis. It is assumed that thermal effects are weakly coupled with the system dynamics behavior, i.e. that the thermal loads can be computed a priori without direct regard for the dynamic changes in confguration. A possible future extension is to provide increased coupling, i.e. to allow periodic updating of the timewise varying thermal loading as a function of the configuration. Although the primary purpose of the Solution Path IV is to handle the effects of quasi-static thermal loadings, the requirements for handling quasi-static mechanical loadings are virtually identical, and the approach allows both types of loading to be analyzed together.

General Approach - The approach utilizes three technical modules - a thermal analyzer, a structural statics and dynamics analyzer such as NASTRAN, and the DISCOS system-dynamics analyzer. The approach requires a computation of nodal temperatures which define the timewise varying thermal loads; a definition and selection of appropriate structural mode shapes; and a determination of the timewise varying thermal deformations expressed in terms of modal amplitudes. These modal thermal deformations can then be used by DISCOS as quasi-static loads within a time-domain system-dynamics analysis.

Definitions - The following primary symbols are used.

- q = vector of nodal displacements
- Q = vector of nodal forces
- a = vector of structural mode amplitudes
- A = vector of generalized modal forces
- K^n = nodal stiffness matrix ($Q = K^n q$)
- K^m = modal stiffness matrix ($A = K^m a$)
- ϕ = matrix of mode shapes ($q = \phi a$)

I = the identity matrix

T = transpose operator for matrix or vector

Basic Considerations - By definition, the nodal/modal displacement relationship is

$$q = \phi a \tag{F.1-1}$$

By reciprocity, for a linear elastic system, forces are given by the transpose relationship

$$A = \phi^T Q \tag{F.1-2}$$

By substitution

$$A = \phi^T Q = \phi^T K^n q = (\phi^T K^n \phi)\, a = K^m a$$

where

$$K^m = \phi^T K^n \phi \tag{F.1-3}$$

For the IAC Solution Path IV application, the selected mode shapes ϕ might take any of several forms:

1) Dynamics modes, which are orthogonal or orthonormal with respect to the nodal mass or stiffness matrix.

2) Selected non-orthogonal thermal deformation shapes.

3) Arbitrary user defined mode shapes.

4) Some combination of (1), (2) and (3).

The selection criterion is that the modes must allow an adequate simulation, within practical limits, of the actual time history deformations, via a DISCOS time domain analysis.

Conceptual Procedure - Given a selected set of mode shapes, Φ, the computational steps for a body in the system can be defined in the following conceptual manner.

1) Compute the nodal stiffness matrix, K^n, for the body.

2) Compute the modal stiffness via Equation F.1-3, $K^m = \Phi^T K^n \Phi$.

3) Apply the thermal loading for the first time point, and, with the structure "locked" (all nodal displacements prevented), compute the resulting nodal loads, -Q. (Q could be supplemented at this point, if desired, by additional mechanical loadings).

4) Compute the equivalent generalized modal loads via Equation F.1-2, $A = \Phi^T Q$.

5) Solve for modal displacements, using

$$a = (K^m)^{-1} A \qquad \text{(F.1-4)}$$

6) Repeat steps 3-5 at each time point for which thermal displacements are required.

7) Provide DISCOS with the matrix of thermal amplitudes, a, versus time, for use as quasi-static displacement loadings.

F.2 IMPLEMENTATION

IAC Procedure - A solution could be obtained by following, in a direct fashion, the steps of the above described conceptual procedure. These steps might be accomplished via an appropriate data handling sequence within a standard structural statics analyzer, e.g. via NASTRAN DMAP.

However, a direct and more automatic procedure for handling this type of solution is already available within standard structural analyzers such as NASTRAN, via MPC equations. The modal amplitudes, a, are first introduced as additional "scalar freedoms." All nodal freedoms are then constrained to follow the mode shapes, via the MPC equations

$$q = \phi a \tag{F.2-1}$$

Using the input thermal loading at each time point as a load case, NASTRAN will automatically perform the equivalent of the conceptual steps 1-6, described in Section F.1, resulting in the required matrix of modal amplitudes versus time. This procedure involves decomposition of the modal stiffness matrix, K^m. The size of K^m is generally much smaller (and therefore the decomposition cost much less) than that of the nodal stiffness, K^n.

The IAC Solution Path IV includes automatic generation of required NASTRAN input data, i.e. creation of the MPC equations from the selected structural mode shapes, and creation of the thermal load sets from the selected nodal-temperature-versus-time array.

<u>Alternative Procedures</u> - As an aid to understanding of the concepts, it is useful to consider several alternative ways in which the just described conceptual procedure can be viewed. The basic relationships in these procedures are still the same.

One procedure arises from noting that Equation F.1-1 defines nodal displacements in terms of modal displacements, but that the required modal amplitude for DISCOS might be determined using an "inverse" relationship. The mode shape matrix ϕ is generally rectangular and cannot be inverted directly. However, the following relationship can be derived.

$$\begin{aligned} a = (K^m)^{-1}A &= (\phi^T K^n \phi)^{-1}(\phi^T Q) \\ &= (\phi^T K^n \phi)^{-1}(\phi^T K^n q) \equiv Pq \end{aligned} \tag{F.2-2}$$

where the matrix P may be regarded as a "pseudo inverse" of the ϕ matrix. This equation may be viewed as a minimum-potential-energy formulation subject to the constraints $q = \phi a$. To utilize the pseudo inverse matrix, one could compute nodal displacements, q, via $q = (K^n)^{-1}Q$, and then determine modal amplitude, a, via the psuedo inverse. This, however, is computationally disadvantageous, as it is evident that the vector q would be computed unnecessarily, and essentially the operation $(K^n)(K^n)^{-1}$ would be performed.

Another viewpoint is that the required modal amplitudes should be computable via an "inner product" operation of the form

$$a = \phi^T K^n q \tag{F.2-3a}$$

This again requires an explicit determination of the vector q, and further requires that the modes be orthonormal with respect to the nodal stiffness, K^n. In fact, Equation (F.2-3a) is just a special case of Equation (F.2-2), where, because of the orthonormality condition,

$$\phi^T K^n \phi = I \tag{F.2-3b}$$

A simpler form of the pseudo inverse can be derived, and the resulting expression is what has also been called the general "least squares" relationship. Given a vector, q, of nodal quantities, which is defined as the product of a matrix, ϕ, times a vector, a, of modal quantities (i.e. $q = \phi a$), then a reverse relationship is derivable as

$$a = Ia = (\phi^T\phi)^{-1}(\phi^T\phi)a = (\phi^T\phi)^{-1}\phi^T q \tag{F.2-4}$$

This again requires an explicit determination of q, via an inversion or decomposition of K^n. However, it is not required that the modes be orthonormal, only that the matrix $(\phi^T\phi)$ have an inverse. The calculation for a via Equation F.2-4 is exactly equivalent to that of Equation F.2-2 only if q has been computed by a minimum potential energy solution using K^n, and subject to the constraints $q = \phi a$. If q does not satisfy $q = \phi a$ (in general

it does not), a is computed such that the sum of squares of the elements of q is minimized. Thus a "good" least squares solution, instead of an "optimum" minimum potential energy solution, is obtained. It is interesting to note that Equation F.2-4 would result via a direct and obvious substitution from Equation F.2-2, for the case where K^n is an identity matrix (i.e. the unit diagonal least squares coefficients).

Several of the concepts discussed above are based upon the viewpoint that the modes will be made orthonormal. A set of non-orthonormal modes, ϕ, may be converted to an orthonormal set, $\hat{\phi}$. The orthonormality condition is given by

$$\hat{K}^m = \hat{\phi}^T K^n \hat{\phi} = I \tag{F.2-5}$$

For non-orthonormal modes, the modal stiffness, K^m, will not be an identity matrix, but must be positive definite. Orthonormalization involves computing a transformation matrix, X, such that

$$\hat{\phi} = \phi X \tag{F.2-6a}$$

and

$$\hat{K}^m = X^T K^m X = I \tag{F.2-6b}$$

Additional cost is required for orthonormalization. However, it simplifies some of the required computational steps, for example, the solution step represented by Equation (F-4) is eliminated, since

$$a = (\hat{K}^m)^{-1} A = IA = A \tag{F.2-7}$$

F.3 EXAMPLE

In order to describe in more concrete terms some of the methodology involved, the simple example problem of a cantilever beam is used. The beam is sketched in Figure F.3-1 with its associated X-Y coordinate system.

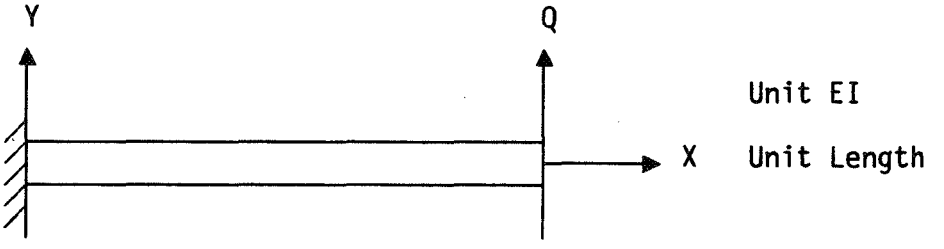


Figure F.3-1: Solution Path IV Cantilever Beam Example

Problem Definition - In order to avoid unnecessary complexity, the beam is considered to have unit EI and length. The thermal loading considered is such that, if the beam were permitted to deform, the thermally deflected shape would be the same as that of a cantilever under a unit concentrated end load, Q. Thus step 3 in the previously described conceptual procedure (Section F.1) becomes trivial. A final simplification is that the mode shapes and stiffness matrix are developed based on continuous beam theory, rather than on a finite element nodal discretization. This simplified example does, however, preserve the essential ingredients of the general Solution Path IV problem.

Mode Shapes - The moment along the beam due to thermal load, is given by

$$Moment = Q\,(1 - X) \tag{F.3-1a}$$

The slope is

$$Slope = \int_0^X Q\,(1 - X)\,dX = Q\left(X - \frac{X^2}{2}\right) \tag{F.3-1b}$$

and the deflection is

$$Deflection = \int_0^X Q\left(X - \frac{X^2}{2}\right)dX = Q\left(\frac{X^2}{2} - \frac{X^3}{6}\right) \tag{F.3-1c}$$

For our purposes, three structural mode shapes (0, 1, 2) will be defined, as follows. (End deflections are shown in parentheses.)

$$Y = \phi_0(X) = 15X^2 - 5X^3 (\Delta_{X=1} = 10) \tag{F.3-2a}$$

$$Y = \phi_1(X) = 16X^2 - 5X^3 (\Delta_{X=1} = 11) \tag{F.3-2b}$$

$$Y = \phi_2(X) = 14X^2 - 5X^3 (\Delta_{X=1} = 9) \tag{F.3-2c}$$

Mode 0 is capable of exactly representing the deflected shape under end load. Modes 1 and 2 are mutually independent but non-orthogonal modes, which can be combined to exactly represent mode 0. Curvatures are given by

$$Y'' = \phi_0''(X) = 30 - 30X \tag{F.3-3a}$$

$$Y'' = \phi_1''(X) = 32 - 30X \tag{F.3-3b}$$

$$Y'' = \phi_2''(X) = 28 - 30X \tag{F.3-3c}$$

Modal Stiffnesses - In the general Solution Path IV problem, the modal stiffness matrix is defined in terms of the discretized nodal matrix, via Equation F.1-3. In the present simplified example, however, the modal stiffnesses are defined via the integral

$$K_{ij}^m = \int_0^1 \phi_i''(X) \cdot \phi_j''(X)\, dX \tag{F.3-4}$$

For example, the stiffness for mode 0 is

$$K_{00}^{m} = \int_0^1 (30 - 30X)^2 \, dx = 300 \tag{F.3-5}$$

Considering only modes 1 and 2, the modal stiffness is

$$K^m = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} = \begin{bmatrix} 364 & 296 \\ 296 & 244 \end{bmatrix} \tag{F.3-6a}$$

The inverse of this two-mode stiffness matrix is

$$(K^m)^{-1} = \frac{\begin{bmatrix} 244 & -296 \\ -296 & 364 \end{bmatrix}}{1200} = \begin{bmatrix} .203333 & -.246667 \\ -.246667 & .303333 \end{bmatrix} \tag{F.3-6b}$$

Modal Loads - For the simple thermal loading considered (Q=1), the modal loads are given by the general form

$$A_i = \phi_i(X=1) \cdot (Q=1) \tag{F.3-7}$$

For the three modes considered here

$$A_0 = 10 \cdot 1 = 10 \tag{F.3-8a}$$

$$A_1 = 11 \cdot 1 = 11 \tag{F.3-8b}$$

$$A_2 = 9 \cdot 1 = 9 \tag{F.3-8c}$$

Amplitudes and Displacements - For the case of the unit concentrated end load, the exact (simple beam theory) value for vertical end displacement is

$$\Delta = \frac{1}{3} \tag{F.3-9}$$

Using only mode 0, the modal amplitude is

$$a_0 = K_{00}^{-1} A_0 = (300)^{-1}(10) = .033333 \tag{F.3-10a}$$

and the end displacement is

$$\Delta = (a_0)(10) = .333333 \tag{F.3-10b}$$

As expected, this displacement agrees with the exact value, since mode 0 is, in fact, the correct deflected shape for a cantilever under end load.

Using only mode 1, the modal amplitude is

$$a_1 = K_{11}^{-1} = (364)^{-1}(11) = .030220 \tag{F.3-11a}$$

and the end displacement is

$$\Delta = (a_1)(11) = .332418 \tag{F.3-11b}$$

Note that this displacement differs from the correct value by only about .3%, even though the shape differs from the true shape by roughly 5%.

Similarly, using only mode 2, the modal amplitude is

$$a_2 = K_{22}^{-1} A_2 = (244)^{-1}(9) = .036885 \tag{F.3-12a}$$

and the end displacement is

$$\Delta = (a_2)(9) = .331967 \tag{F.3-12b}$$

Using modes 1 and 2 in combination, the modal amplitudes are

$$\begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} = \begin{bmatrix} .203333 & -.246667 \\ -.246667 & .303333 \end{bmatrix} \begin{Bmatrix} 11 \\ 9 \end{Bmatrix} = \begin{Bmatrix} .016667 \\ .016667 \end{Bmatrix} \tag{F.3-13a}$$

and the end displacement is

$$\Delta = \begin{Bmatrix} .016667 \\ .016667 \end{Bmatrix} \begin{Bmatrix} 11 \\ 9 \end{Bmatrix} = .333333 \qquad \text{(F.3-13b)}$$

Note that these two modes in combination produce the exact end displacement, with each of the modes contributing approximately one-half its value.

F.4 CONCLUSIONS

Truncation Considerations - In comparing the results of Equations F.3-11 to F.3-13, an important fact to note is that very different modal amplitudes are computed, depending upon which modes are allowed to participate in the solution. E.g. when mode 1 is used alone, its computed amplitude in Equation F.3-11a is .030220; however, when modes 1 and 2 are used in combination, the computed amplitude for mode 1 in Equation F.3-13a is only .016667. Obviously it would not be valid to allow both of these modes to participate in the solution, and then to arbitrarily perform a truncation which discards one of the modes.

This situation occurs because the modes employed are non-orthogonal, and therefore the non-diagonal modal stiffness matrix couples the individual modal effects together. As shown by Equation F.2-6, the modes could be normalized. However, this should not be done for Solution Path IV with the viewpoint of allowing for a post-amplitude-solution truncation, because the mode shapes at that point would be expressed via a different (orthogonal) basis from that selected by the user. The user would then not be able to control the truncation in terms of the user selected (presumably the "most important") set of mode shapes. The IAC implemented methodology does allow the user complete freedom to truncate the solution process via initial selection of the mode shapes.

Solution Path V Applicability - It can be noted that the IAC methodology for Solution Path IV would also appear to provide an effective implementation for part of the fully coupled thermal/structural/control Solution Path V capability. A "thermal mode" (nodal temperature vector) in the Solution Path V can be treated in the same manner as a thermal loading at a particular time point in Solution Path IV. The structural modes in the two solution paths are similar, although these modes in Solution Path V will probably be normalized with respect to the mass and stiffness matrices. In Solution Path V, it will probably be desirable to truncate both the structural modes and the thermal modes, in order to obtain a solution which is computationally efficient but still meaningful to the user.

-3.0 BOEING PROPRIETARY MODULES

The execution of Boeing proprietary modules within IAC is handled in the same manner as for the other modules discussed in Section 3. The following sections define the requirements and options for executing each of the currently available Boeing proprietary IAC modules.

-3.1 EASY5/EZPLOT

The EASY5 system is a set of Boeing proprietary general purpose programs that can be used to analyze a wide range of continuous or discrete nonlinear systems. This section describes the IAC implementation of EASY5, and the associated graphics module EZPLOT. The general RUN schematic for executing these modules within ACE is shown in Figure -3.1-1. The following is a summary of EASY5 RUN parameters which may be specified.

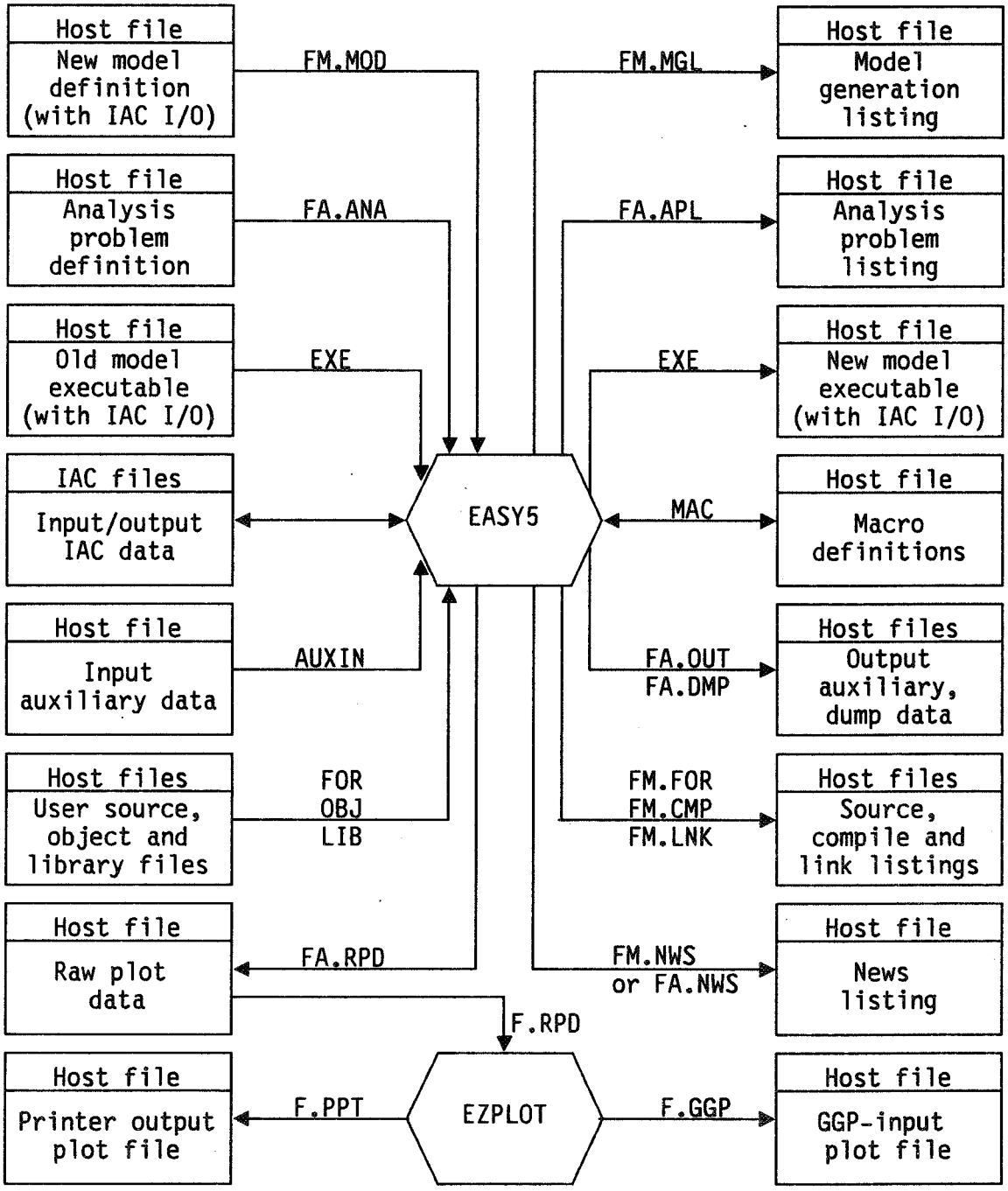
FM	=	basic model file name	
FA	=	basic analysis file name	
PRT	=	print option	(default = NO)
EXE	=	executable load module spec	
FOR	=	user source specs	
OBJ	=	user object specs	
LIB	=	user library specs	
LIST	=	listing generation options	(default = NOFOR, NOCMP, NOLNK, NONWS)
SCRATCH	=	scratch directory spec	
AUXIN	=	auxiliary input spec	
MAC	=	macro definitions spec	

A common abbreviated form of the EASY5 RUN statement is

```
RUN EASY5 (FM = name, FA = name)
```

This statement executes a two-part EASY5 module, consisting of a model processor to generate an executable program, and the program execution to perform an analysis.

The FM parameter names basic model EASY files. The new model definition file input by the user must have type MOD; MGL is the model generation listing file; FOR, CMP and LNK are the respective FORTRAN, compiler and linker output files; and NWS is the output news file. If the FM parameter is not given, the FA parameter is used to name the news file, and the other file types are not used.




```
RUN EASY5  (FM=name, FA=name, PRT=option, EXE=spec+
           ,FOR=(spec-list), OBJ=(spec-list), LIB=(spec-list)+
           ,LIST=(option-list), SCRATCH=spec, AUXIN=spec, MAC=spec)

RUN EZPLOT (F=name, PRT=option, TYPE=(type-list))
```

Figure -3.1-1: EASY5/EZPLOT RUN Schematic

FA names basic analysis EASY files, and is required if FM is not given. The analysis problem definition file input by the user must have type ANA; APL is the analysis problem listing file; OUT is the output auxiliary data file; DMP is the output dump data file; RPD is the output raw plot data file; and NWS is the output news file, if the FM parameter is not given.

The complete set of EASY5 parameters is given by the RUN statement shown in Figure -3.1-1.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default) or HOLD: YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

EXE gives the spec of the executable load module, to be run during the analysis. If FM is not given, EXE is required and gives the spec of an executable created from a previous model. If FM is given, EXE is optional and gives the spec of the executable to be saved; in this case, if EXE is not given, the executable is deleted.

FOR, OBJ and LIB identify user subroutines in the form of source code, object code and object libraries, respectively (the source and object routines may be identified by file names and/or specs). These subroutines are linked into the EASY5 module prior to execution; in case of duplicate subroutine names, user routines will replace the corresponding EASY5 routines. Note that each value of LIB defines a complete link type of spec which may include the directory and type and must include "/LIB".

LIST controls generation of the model processor and news output listings (printing is controlled by PRT); it is a keyvalue list which may contain FOR or FOR, CMP or NOCMP, LNK or NOLNK, and NWS or NONWS, to specify generation or no generation of the respective FORTRAN, compiler, linker and news output files (the appropriate files are generated in case errors are detected, regardless of the LIST values).

SCRATCH allows the user to specify a host directory for storing scratch files during EASY5 execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

AUXIN may be used to give the spec of an auxilliary input file, whose data is referenced by the analysis problem definition. Note that this file may be the FA.OUT file created during a previous run.

MAC may be used to give the spec of a file where EASY input or output macro definitions are stored. If the file already exists, its macros may be input to the model definition, and any newly defined macros may be output and added to the file.

EZPLOT Execution - EZPLOT provides batch and/or interactive display of plots from an EASY5 generated plot file. The following is a summary of EZPLOT RUN parameters which may be specified.

F	=	basic file name	
PRT	=	print option	(default = NO)
TYPE	=	output plot file types	

The F parameter names basic EZPLOT files. File type RPD is the input raw plot data file, generated by EASY5; PPT is the printer output plot file; and GGP is the output plot file for input to the Boeing GGP plot package.

PRT controls printing of the printer output plot file, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the file unprinted in the directory; and HOLD causes it to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

TYPE specifies a list of desired plot types, which may include one or more of the values PRINTER, GGP and ONLINE. PRINTER requests a printer plot; GGP

requests creation of GGP-input plot file; and ONLINE requests that online (interactive) plotting activity be initiated.

-3.2 FEMNET

FEMNET converts a FEM (Finite Element Model) to a NET (lumped nodal NETwork, i.e. finite difference, model). The FEM is defined via a file in SUPERTAB universal format. The NET may be generated consistent with the bulk input requirements of the SINDA and/or BETA thermal analysis module(s). FEMNET may also be used to generate the FEM and/or NET nodal coordinate meshes in IAC relation form, e.g. for later use by the MIMIC module; and the NET mesh in SUPERTAB universal form, e.g. for a graphical display superimposed on the FEM model.

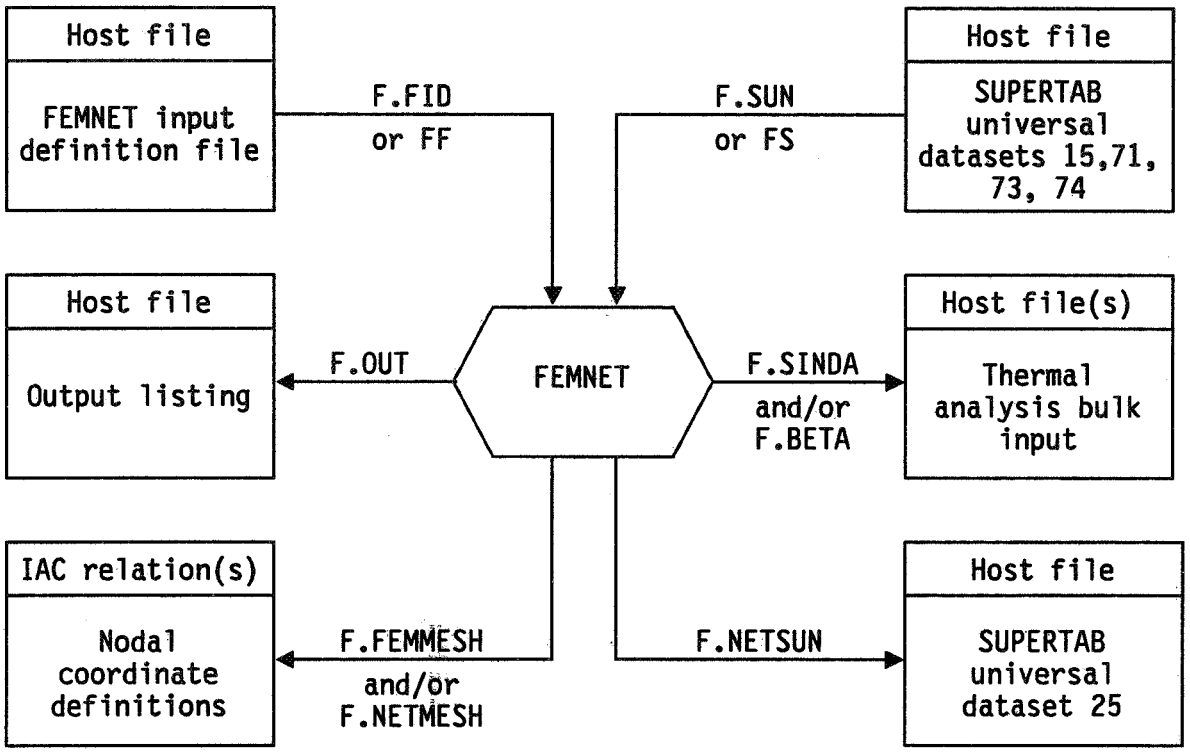
This section describes the IAC implementation of the FEMNET module. Detailed FEMNET documentation is contained in Reference -2. The general RUN schematic for executing FEMNET within ACE is shown in Figure -3.2-1. The following is a summary of FEMNET RUN parameters which may be specified.

F	=	basic file name	
FF	=	FEMNET input definition file spec	
FS	=	input SUPERTAB universal file spec	
TYPE	=	output file types	(default = SINDA)
PRT	=	print option	(default = NO)
EXE	=	user load module spec	

The F parameter in the RUN statement is a basic file root name, used to identify input files and to name output files. File type FID is the FEMNET input definition file; SUN is the input SUPERTAB universal file; OUT is the output listing file; SINDA and BETA are the output thermal analysis files for input to the respective SINDA and BETA modules; FEMMESH and NETMESH are output IAC relation files defining the respective FEM and NET meshes; and NETSUN is the output SUPERTAB file defining the NET mesh.

FF and FS are optional parameters; if given they define complete input file specs to be used instead of the respective F-specs F.FID and/or F.SUN.

TYPE selects the file types (SINDA, BETA, FEMMESH, NETMESH, NETSUN) to be output. Default is the single file of type SINDA.




```
RUN FEMNET (F=name, FS=spec, FF=spec, TYPE=(type-list)+
           ,PRT=option, EXE=spec)
```

Note - The TYPE list may contain (SINDA, BETA, FEMMESH, NETMESH, NETSUN).

Figure -3.2-1: FEMNET RUN Schematic

The PRT parameter controls printing of the output listing file, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the file unprinted in the directory; and HOLD causes it to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

EXE is an optional parameter; if given it defines a complete spec for a user-defined load module, to be executed instead of the standard FEMNET module.

-3.3 BETA/IBET

This section describes the IAC implementation of the BETA module (Boeing Engineering Thermal Analyzer), and the associated IAC database interface module IBET (Interface from BETa). BETA is based on a nodal network (finite difference) approach and can include the efects of conduction, convection, radiation, and variable thermal properties on transient or steady-state temperature distributions. Detailed BETA documentation is contained in Reference -3. The general RUN schematic for executing these modules within ACE is shown in Figure -3.3-1. The following is a summary of BETA RUN parameters which may be specified.

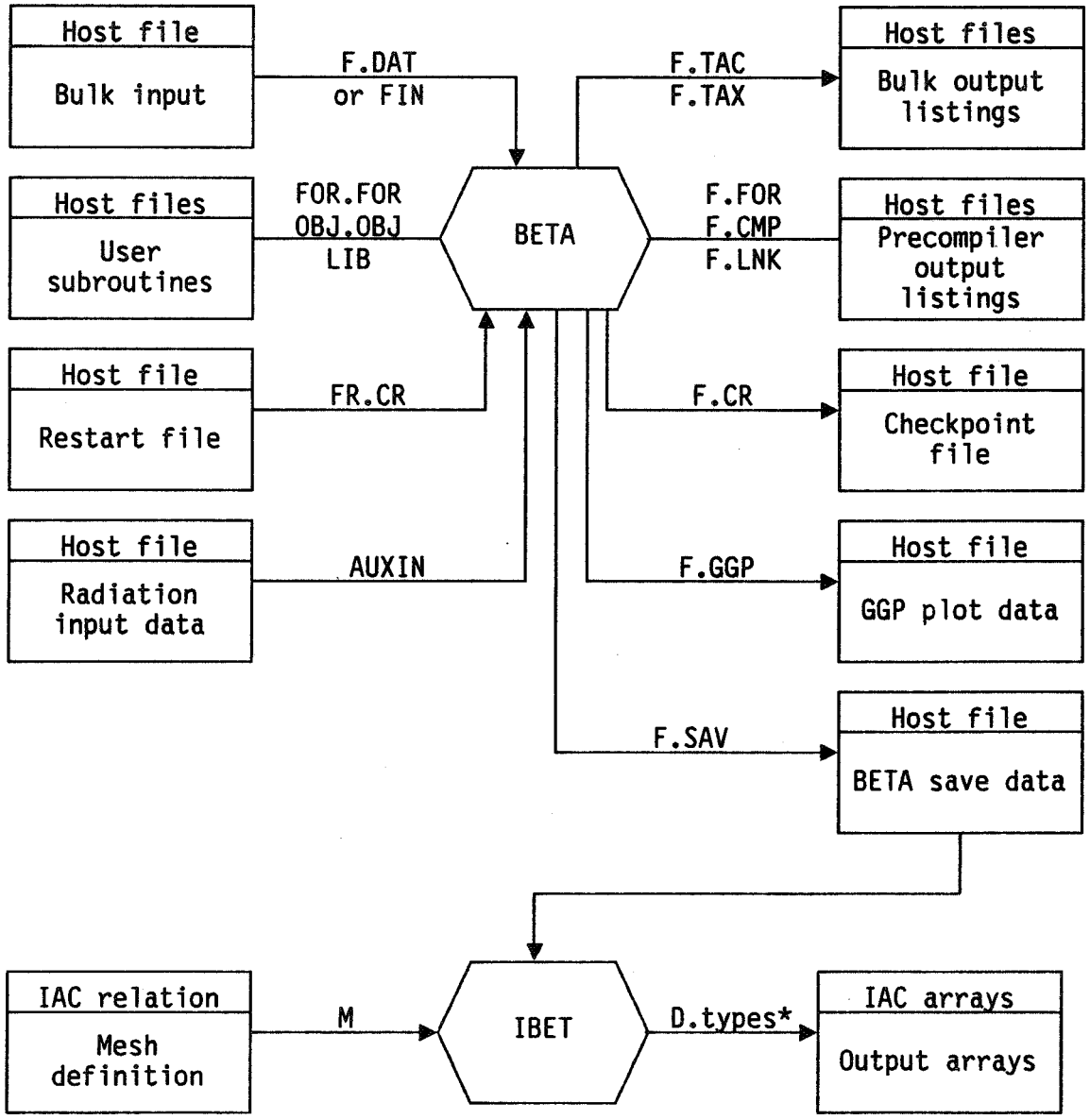
```
F        = basic file name
FIN      = input file spec
PRT      = print option                 (default = NO)
INTERVAL = save time interval
FOR      = user source names
OBJ      = user object names
LIB      = user library specs
LIST     = listing generation options   (default = NOFOR,NOCMP,NOLNK)
SCRATCH  = scratch directory spec       (default = installation defined)
FR       = restart file name
AUXIN    = auxiliary input spec
```

A common abbreviated form of the BETA RUN statement is

```
RUN BETA (F = name, PRT = option)
```

This statement executes a two-part BETA module, consisting of a preprocessor to generate an executable processor program, and the processor to perform the analysis.

The F parameter in the RUN statement is used to name basic BETA files. As shown in Figure -3.3-1, file type DAT is the bulk input file; TAC and TAX are output listing files; SAV is an optional output file containing data which may be inserted into an IAC database; FOR, CMP and LNK are the respective



*types = TEMP, NCAP, COND, QFLOW, QAPPL, QNET

```
RUN BETA  (F=name, FIN=spec, PRT=option, INTERVAL=interval+
          ,FOR=(name-list), OBJ=(name-list), LIB=(spec-list)+
          ,LIST=(option-list), SCRATCH=spec, FR=name, AUXIN=spec)

RUN IBET  (FB=spec, M=spec, D=name, TYPE=(type-list))
```

Figure -3.3-1: BETA/IBET RUN Schematic

FORTRAN, compiler and linker output files; CR is the checkpoint file (i.e. BETA's saved problem tape); and GGP is the output plot file for input to the Boeing GGP plot package.

The PRT parameter controls printing of the output listing files, and equals one of the keyvalues YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

The complete set of BETA parameters is given by the RUN statement shown in Figure -3.3-1.

FIN is an optional parameter; if given it defines a complete input file spec to be used instead of the F-spec name.DAT.

INTERVAL is an optional parameter; if given it causes output of certain BETA computed data to the binary F.SAV file, and specifies the time interval at which that data is saved. The interval is relative to the BETA print time step, e.g. a value of 1 indicates data saved at each print time step, 2 indicates every other print time step, etc. Selected data from the save file can later be processed and stored in an IAC database, via the interface module IBET.

FOR, OBJ and LIB identify user subroutines in the form of source code, object code and object libraries, respectively. These subroutines are linked into the BETA module prior to execution; in case of duplicate subroutine names, user routines will replace the corresponding BETA routines. Note that each value of LIB defines a complete link type of spec which may include the directory and type and must include "/LIB".

LIST controls generation of the preprocessor output listings (printing is controlled by PRT); it is a keyvalue list which may contain FOR or NOFOR, CMP or NOCMP, and LNK or NOLNK, to specify generation or no generation of the respective FORTRAN, compiler and linker output files (compiler and linker

files are generated in case errors are detected, regardless of the LIST values).

SCRATCH allows the user to specify a host directory for storing scratch files during BETA execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

The FR parameter gives the name (or optionally the entire leading portion of a file spec, down through the name) for restarting from a BETA saved problem tape. BETA refers to restarting as a recall operation. The restart file contains the problem description (i.e. the generated XTRNL1, XTRNL2, etc. subroutines from the preprocessor) and partial results from the processor. When restarting, the preprocessor calls subroutine RECALL to copy the old problem description to an intermediate file called IEDITP (unit 13). The preprocessor then merges the old and restart problem description on unit 14. The contents of unit 14 as well as any required user subroutines are compiled and linked to form the processor for restarting. All user subroutines input for the checkpoint run, must be input for the restart run. Subroutine XTRNL8 of the processor will read the partial results.

AUXIN may be used to give the spec of an optional radiation input file.

<u>BETA/IAC Data-Flow Approach</u> - In developing the BETA/IAC interface approach, a major consideration is the fact that the BETA bulk input file may contain general user-supplied FORTRAN-like statements, which are compiled into executable code by a BETA preprocessor. It is also considered desirable to have an open-ended BETA/IAC data-flow approach, whereby additional data may in the future be identified for storage in the database. The developed approach therefore consists of the following steps.

1) BETA is run by the user. The normal BETA precompiler generates appropriate FORTRAN code, based on the user's bulk input file.

2) If the INTERVAL parameter has been specified, an IAC processor modifies the BETA generated FORTRAN by inserting calls to a utility routine

IACSAV. Each call sequence identifies a BETA variable name (e.g. T) representing a vector of data, and a corresponding IBET nametag, to be saved on the F.SAV file.

3) The BETA link and execution steps are performed.

4) The interface routine IBET may be run by the user, with a list of selected IAC array types. IBET finds on the binary F.SAV file all nametag data variables required to create the selected arrays, creates the arrays, and inserts them into the IAC database.

This approach uses existing BETA data handling tools and avoids modification of the BETA code. It also provides a general capability by which additional BETA/IAC data-flow can be implemented in the future.

IBET Execution - The following is a summary of IBET RUN parameters which may be specified.

FB	= BETA save file spec	
M	= mesh relation spec	
D	= output array name	
TYPE	= output array types	(default = TEMP)

The form of the IBET RUN statement is

RUN IBET (FB = spec, M = spec, D = name, TYPE = (type-list))

The FB parameter gives the spec for the save file of BETA computed data, to be input to IBET.

M is an optional parameter which gives the spec of a mesh relation (e.g. the NETMESH file from FEMNET) to be read from the database. If M is provided, the attributes (e.g. NODE,X,Y,Z) in this relation are incorporated, where appropriate, into the BETA arrays output to the database.

D gives the name (or optionally the name:number) for all array files to be output by IBET to the IAC database.

TYPE selects the array file types (TEMP, NCAP, COND, QFLOW, QAPPL, QNET) to be output to the IAC database. Default is the single array of type TEMP.

BETA/IAC Arrays - The six currently available IAC array file types are described in Figure -3.3-2. Note that if the M parameter is specified, all occurrences of the attribute NID in the figure are replaced by the M mesh relation attributes.

An example RUN statement for IBET might be

```
RUN IBET (FB = BETPROB.SAV, D=PROBX, TYPE = (TEMP, QFLOW))
```

IBET would then read the BETA save file BETPROB.SAV, and create and insert the array files PROBX.TEMP and PROBX.QFLOW, into the IAC database.

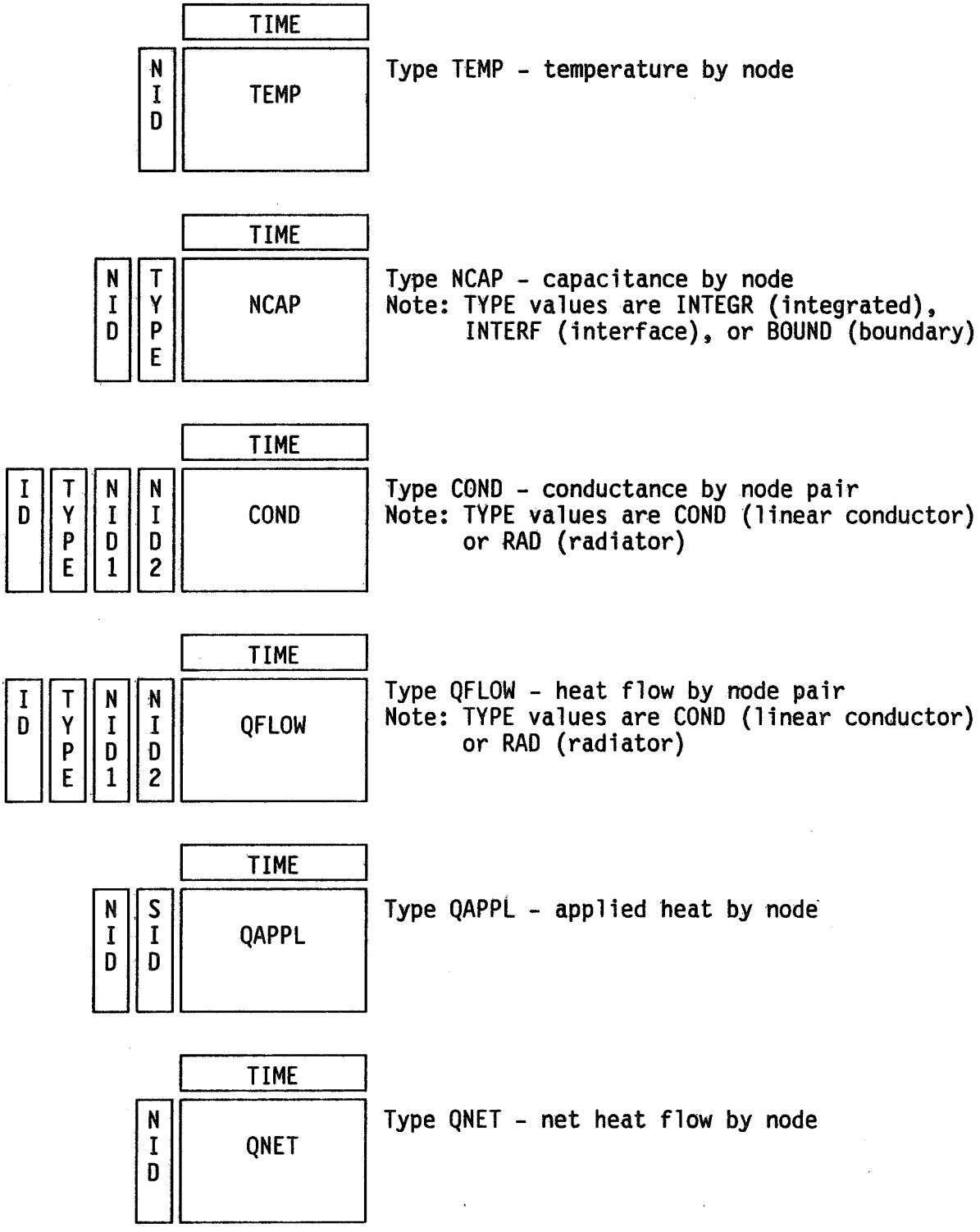


Figure -3.3-2: BETA/IAC Arrays

Array type/description	Attribute name/type/description
TEMP - temperature by node	TEMP R1 - temperature values NID I1 - node id's TIME R1 - time values
NCAP - capacitance by node	NCAP R1 - capacitance values NID I1 - node id's TYPE C6 - node types TIME R1 - time values
COND - conductance by node pair	COND R1 - conductance values ID I1 - connector id's TYPE C4 - connector types NID1 I1 - node id 1 NID2 I1 - node id 2 TIME R1 - time values
QFLOW - heat flow by node pair	QFLOW R1 - heat flow values ID I1 - connector id's TYPE C4 - connector types NID1 I1 - node id 1 NID2 I1 - node id 2 TIME R1 - time values
QAPPL - applied heat by node	QAPPL R1 - applied heat values NID I1 - node id's SID I1 - source id's TIME R1 - time values
QNET - net heat flow by node	QNET R1 - net heat flow values NID I1 - node id's TIME R1 - time values

Figure -3.3-2: BETA/IAC Arrays - (Continued)

-3.4 COPM

COPM (Concentrator Optical Performance Model) is an analytical code for performance calculations on a parabolic dish solar concentrator. COPM can analyze both single reflector and cassegrainion optical configurations. The code is based on a Monte-Carlo ray tracing technique.

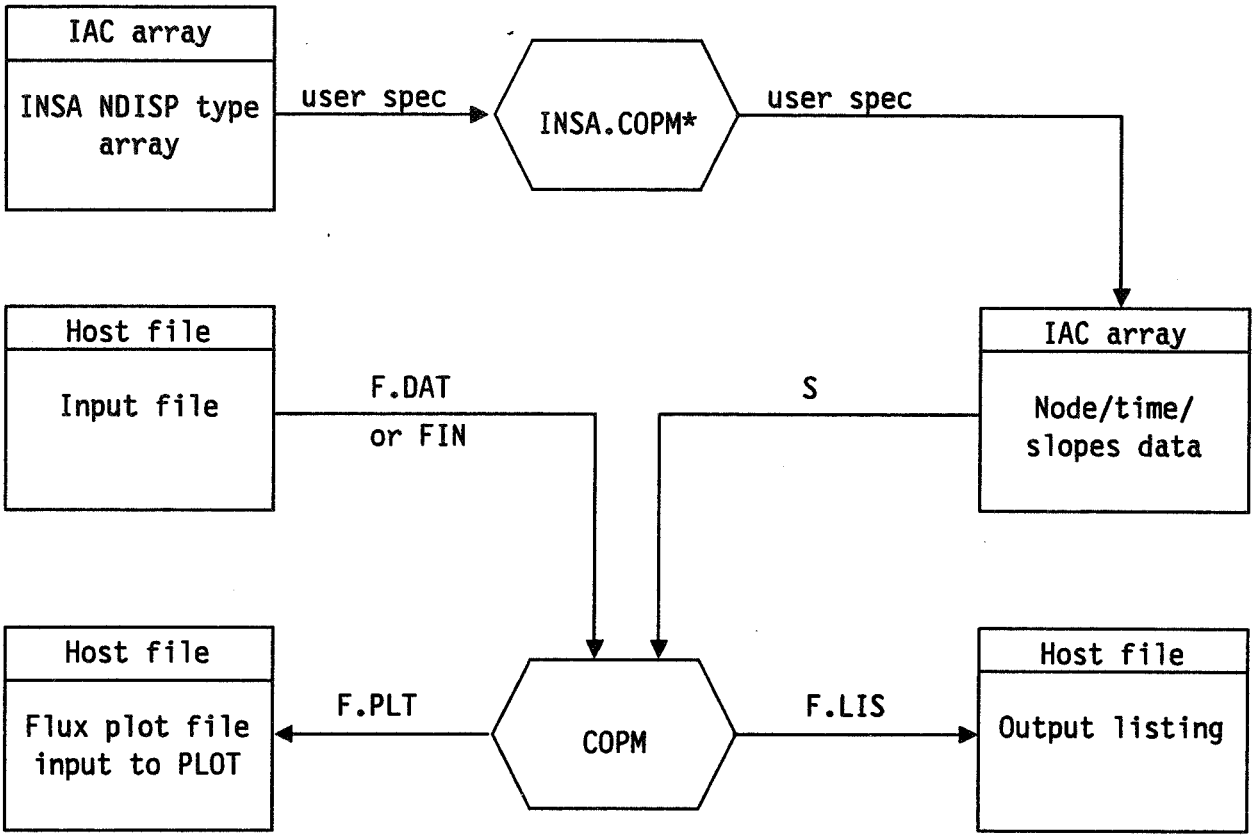
This section describes the IAC implementation of the COPM module. Detailed documentation is contained in References -4 through -6. The general RUN schematic for executing COPM within ACE is shown in Figure -3.4-1. The following is a summary of COPM RUN parameters which may be specified.

F	=	basic file name	
FIN	=	input file spec	
S	=	input slopes array spec	
AREA	=	data storage area	(default = HD)
PRT	=	print option	(default = NO)
DEBUG	=	debug option	(default = 0)
EXE	=	user load module spec	
PLOT	=	Plot id values	

The F parameter in the RUN statement is a basic file root name, used to identify the input file and name the output file. File type DAT is the COPM bulk input file; and LIS is the output listing file.

FIN is an optional parameter; if given, it defines a complete input file spec to be used instead of the F-spec name.DAT.

S is an optional parameter; if given, it defines the spec of an IAC array containing X and Y slope values at node points in the structural mesh. COPM superimposes these slopes upon the surface imperfection effects due to manufacturing errors (input RMS value) and surface specularity. The S array slopes may represent thermal distortion effects (see NDISP array output by module INSA), possibly merged with additional manufacturing error or other slope effects. If S is not given, these slopes are assumed to be zero.



*Note - INSA.COPM is an ACE command file that converts an INSA output array (of nodal displacements) to a COPM input array. The command file is stored on the standard IAC COPM directory.

```
RUN COPM (F=name, FIN=spec, S=spec, AREA=option+
        , PRT=option, DEBUG=option, EXE=spec, PLOT=(time-id-list))
```

Figure -3.4-1: COPM RUN Schematic

0152T/0031-3.4/R5

The AREA parameter gives the storage area from/to which IAC relations and arrays are to be transferred. D indicates database; H indicates host directory; DH indicates database is possible, else host directory; and HD (default) indicates host directory if possible, else database. For example, a COPM array read operation following an AREA=DH specification results in an attempt to retrieve the array from the IAC database; if the retrieval is not successful (e.g. database not open, invalid IAC file specification, or file not found), an attempt is then made to retrieve the array from the host directory.

PRT controls printing of the output listing file, and equals one of the key values YES, NO (default) or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the file unprinted in the directory; and HOLD causes it to be held for the print queue, with later printing by the user followed by automatic deletion from the user's directory.

DEBUG controls the output of certain intermediate results from COPM for problem debugging. The value 0 (default) does not generate debug output, and 1 or 2 generate different sets of output.

EXE is an optional parameter; if given, it defines a complete spec for a user-defined load module, to be executed instead of the standard COPM module.

PLOT specifies a list of time id values for which flux plots are to be created. A zero time id indicates the undeformed (no S array slopes) configuration; and a -n id indicates that only every nth flux point position is to be plotted. PLOT causes COPM to write out a file ready for input into the IAC plot module.

<u>User Subroutines</u> - A user has the option of supplying two routines to handle a dish with non-circular projection.

The first routine, AREAS, calculates the projected area of the dish, ADISH, and an equivalent radius, REQ, where

$$REQ = (ADISH/\pi)^{1/2}$$

The form of the routine is as follows.

```
SUBROUTINE AREAS (ADISH, REQ)
   .
   .
   .
ADISH = ...
REQ = ...
RETURN
END
```

The second routine, INOUT, receives as input arguments the circumferential coordinate, RTH (radians), and the radial coordinate, RD, for a ray point on the primary reflector dish. It determines whether the ray point is inside the perimeter (IRAN returned unchanged) or outside (IRAN returned as 1). If outside, COPM discards the ray and continues to run until the number of rays generated equals NDRAW. The form of the routine is as follows.

```
SUBROUTINE INOUT (RTH, RD, IRAN)
   .
   .
   .
e.g. calculate perimeter radius RP at RTH
IF (RD.GT.RP) IRAN = 1
RETURN
END
```

These routines need to be linked to the rest of the COPM program using the ACE LINK command. This will create a new version of the COPM executable (user defined load module) in the user's host default directory.

-3.7 PLASBE

PLASBE is an interactive module for computing and displaying 1) stable section plastic bending allowables based on the material properties and the section shape factor; and 2) a stress versus total strain curve. The allowables are found from equations developed by Ramberg-Osgood and Cozzone, Reference -8. The computed results can be displayed in table form, or as curves on the terminal, or as hardcopy on an off-line plotter.

The general RUN schematic for executing PLASBE within ACE is shown in Figure -3.7-1. The following is a summary of PLASBE RUN parameters which may be specified.

TERMINAL = device (default = installation defined)
PLOTTER = device (default = installation defined)

A typical form of the RUN statement for PLASBE is

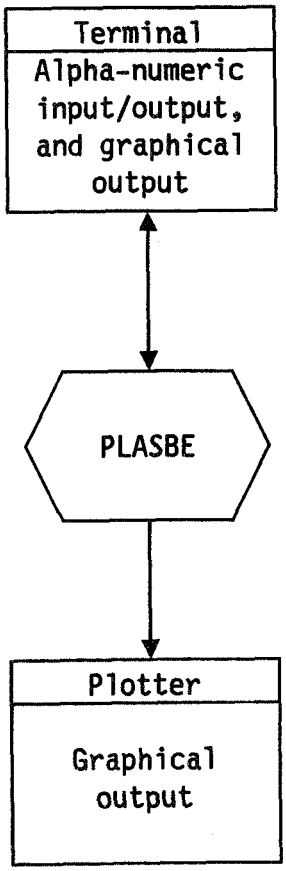
RUN PLASBE (PLOTTER = device)

TERMINAL is an optional parameter for defining the terminal as an on-line graphics device. The on-line ACE help for PLASBE lists the possible devices. The terminal can also be a non-graphics device as long as only tabular output is requested.

PLOTTER is an optional parameter for defining an off-line graphical device. The on-line help lists the possible devices.

PLASBE will prompt for the following input data.

E = Tensile Elastic Modulus
FTU = Tensile Ultimate Stress
FTY = Tensile Yield Stress
UE = Uniform Elongation at FTU
NI = Initial Range Shape Factor
NF = Full Range Shape Factor



RUN PLASBE (TERMINAL=device, PLOTTER=device)

Figure -3.7-1: PLASBE RUN Schematic

0152T/0031-3.7/fig.-3.7-1/R2

FI = Stress Increment

The proper stress-strain curve is dependent on the initial range shape factor (NI) and the full range shape factor (NF). Three conditions exist that depend on the availability of NI and NF.

1. Both NI and NF are available and are input to the program. This is the most accurate representation of the curve. NI is used for calculations between the proportional limit and yield and NF is used for calculations between yield and ultimate stress.

2. NI is available but NF is not. This is the most common case and can be as accurate as the above case. Calculations are carried out exactly as above except that the NF value is calculated from the Ramberg-Osgood two point method. On input, this case is triggered by inputting a zero or blank for NF.

3. Neither NI or NF is available. In this case, the shape factor calculated from the Ramberg-Osgood two point method is used for both NI and NF. This case is triggered by inputting zero or blank for both NI and NF.

After initial data input, a menu is displayed. The menu items are 1) display curves on terminal; 2) display curves on plotter; 3) display table on terminal; and 4) redefine input data. PLASBE always returns to the menu after a menu item is selected and executed. If only a carriage return is entered for a menu item, PLASBE will exit and return to ACE.

-3.8 REDMOD

REDMOD is an interactive module for computing and displaying several of the more commonly used reduced modulus curves for compression and shear (Reference -9). Values from these curves are required for plasticity correction in various structural element expressions. The computed results can be displayed in table form, or as curves on the terminal, or as hardcopy on an off-line plotter.

The general RUN schematic for executing REDMOD within ACE is shown in Figure -3.8-1. The following is a summary of REDMOD RUN parameters which may be specified.

```
TERMINAL = device      (default = installation defined)
PLOTTER = device       (default = installation defined)
```

A typical form of the RUN statement for REDMOD is

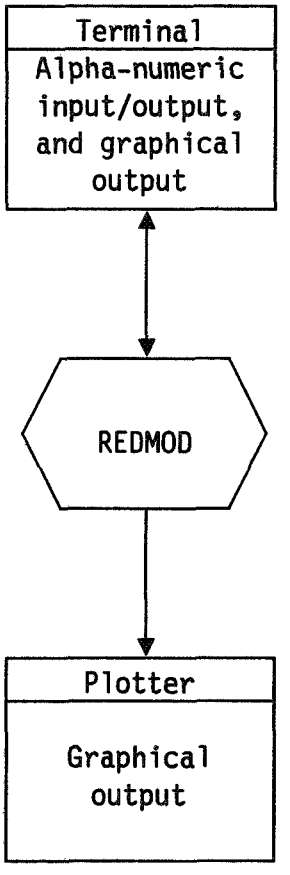
```
RUN REDMOD (PLOTTER = device)
```

TERMINAL is an optional parameter for defining the terminal as an on-line graphics device. The on-line ACE help for REDMOD lists the possible devices. The terminal can also be a non-graphics device as long as only tabular output is requested.

PLOTTER is an optional parameter for defining an off-line graphical device. The on-line help lists the possible devices.

REDMOD will prompt for sets of the following input data.

```
E  =  Elastic Modulus (Compression or Shear)
S  =  Yield Stress
N  =  Shape Factor
I  =  Stress Increment
```



RUN REDMOD (TERMINAL=device, PLOTTER=device)

Figure -3.8-1: REDMOD RUN Schematic

0152T/0031-3.8/fig.-3.8-1/R2

After initial data input, a menu is displayed. The menu allows 1) display curves on terminal; 2) display curves on plotter; 3) display table on terminal; and 4) redefine input data. REDMOD stays in the menu mode until only a carriage return is entered, then a return to ACE is executed.

-3.9 RADSIM/RADINP

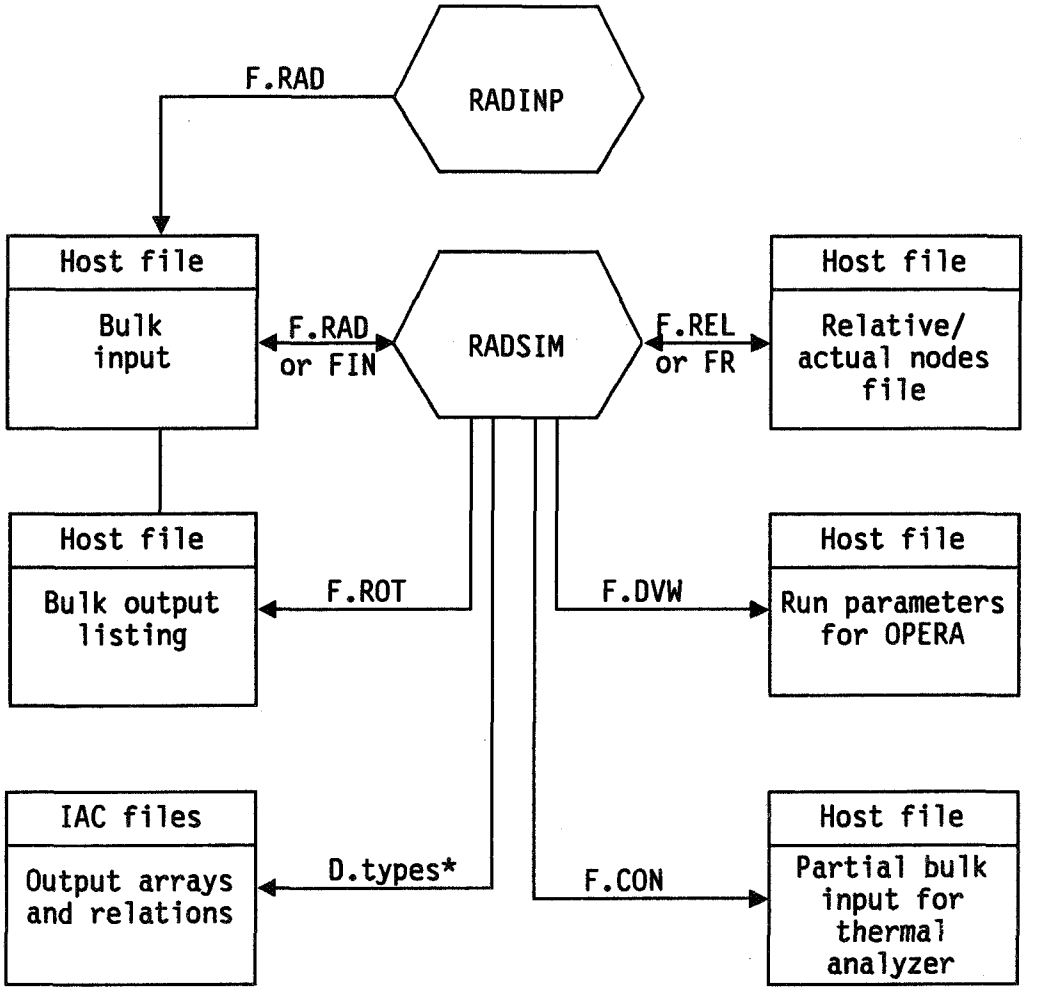
RADSIM (Reference -8) uses a Monte Carlo ray trace technique to calculate radiation exchange factors for complex geometries. The program uses a specular/diffuse reflectance model, can handle adiabatic surfaces, and has the capability of analyzing coupled two wavelength problems. RADSIM can also generate input for various other thermal analyzer modules. The associated module RADINP generates RADSIM input data.

The general RUN schematic for executing these modules within ACE is shown in Figure -3.9-1. The listed RUN statement parameters and diagrammed files describe all of the specific IAC-provided RADSIM capabilities. The following is a summary of RADSIM RUN parameters which may be specified.

F	=	basic file name	
FIN	=	input file spec	
FR	=	relative/actual nodes file spec	
PRT	=	print option	(default = NO)
EXE	=	User load module spec	
TAG	=	input file layout	(default = TAG)
D	=	IAC output file name	
TYPE	=	IAC output file types	(default = ASF,PROP)
AREA	=	IAC data storage area	(default = HD)
MOD	=	Output data modules	(default = OPERA,SINDA)
SYM	=	ASF matrix symmetry option	(default = YES)
GENSYM	=	Symmetry parameters file	(default = standard GENSYM.DAT)
SCRATCH	=	Scratch directory spec	(default = installation defined)

Typical abbreviated forms of the RUN statement are

```
RUN RADSIM (F=name)
RUN RADSIM (F=name, D=name)
```



*Types = ASF,PROP

```
RUN RADINP
RUN RADSIM (F=name, FIN=spec, FR=spec, PRT=option, EXE=spec+
           ,TAG=option, D=name, TYPE=(type-list), AREA=option+
           ,MOD=(module-list), SYM=option, GENSYM=spec, SCRATCH=spec)
```

Figure -3.9-1: RADSIM RUN Schematic

The F parameter in the RUN statement is a basic file root name, used to identify input files and to name output files. File type RAD is the bulk input file containing geometry, surface property and case card data; this file can be generated interactively via the RADINP module. Type REL is an input relative/actual nodes file which relates RADSIM node numbers to thermal analyzer (e.g. SINDA) node id's; the file simply contains a list of node id's, 10 per card (10I8 format). ROT is the bulk output listing file which includes geometry, surface property, view factor, conductor and statistical data. DVW is an output parameter file, to be used as input for the OPERA module. And CON is the output conductor file, to be used as input for a thermal ANALYZER (e.g. SINDA) module.

FIN and FR are optional parameters; if given they define complete input specs, to be used instead of the respective F-specs F.RAD and/or F.REL.

The PRT parameter contols printing of the output listing file and equals one of the key values YES, NO (default), or HOLD; YES causes printing followed by automatic deletion from the user's directory; NO leaves the files unprinted in the directory; and HOLD causes them to be held for the print queue with later printing by the user followed by automatic deletion from the user's directory.

EXE is an optional parameter; if given it defines a complete spec for a user-defined load module to be executed instead of the standard RADSIM module.

TAG has the value TAG (default) or NOTAG, and defines the layout of the bulk input file. TAG indicates nametag/id layout, and NOTAG indicates no-nametag internal sequence number layout.

D names the IAC output files; if not given, these files are not output.

TYPE selects the IAC file types (ASF, PROP) to be output if the D parameter was given. Default is both ASF and PROP.

The AREA parameter gives the storage area to which IAC files are to be transferred. D indicates database; H indicates host directory; DH indicates database if possible, else host directory; and HD (default) indicates host directory if possible, else database. For example, a RADSIM array write operation following the default AREA=DH specification results in an attempt to send the array to the IAC database; if the operation is not successful (e.g. database not open, invalid IAC file spec, or file not found), an attempt is then made to send the array to the host directory.

MOD selects the modules (default is both OPERA and SINDA) for which data is to be generated. Data for OPERA is stored in file F.DVW; and for SINDA in F.CON.

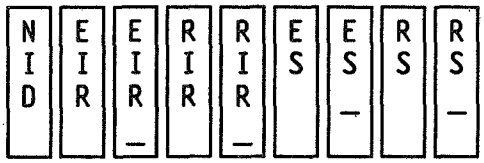
SYM has the value YES (default) or NO, and indicates whether the output ASF matrix is to be made symmetric.

GENSYM allows the user to give the spec for a file containing certain parameters, used to control symmetry and threshold value operations for exchange factor computation. If given, this file is used instead of the default GENSYM.DAT file stored in the standard RADSIM software directory. Note that the standard GENSYM.DAT file contains comments explaining the required format of the file.

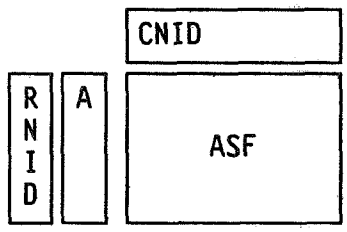
SCRATCH allows the user to specify a host directory for storing scratch files during RADSIM execution, with the default being an installation-dependent scratch directory; the scratch files are automatically deleted after execution is completed.

RADINP Execution - RADINP is an interactive data generation for RADSIM. The spec of the output bulk data file for input to RADSIM is specified by the user via an interactive prompt within the RADINP module. There are no RADINP RUN parameters.

RADSIM/IAC Files - The two currently available IAC file types are described in Figure -3.9-2.



Type PROP - Surface properties by node
Note: Properties are given for positive and negative surfaces, e.g. EIR is for positive surface and EIR_ is for negative surface.



Type ASF - Area times gray body radiation exchange factor

File type/description		Attribute name/type/description	
PROP -	Surface properties by node	NID	I1 - node id's
		EIR	R1 - IR+ emissivities
		EIR_	R1 - IR- emissivities
		RIR	R1 - IR+ reflectivities
		RIR_	R1 - IR- reflectivities
		ES	R1 - solar+ emissivities
		ES_	R1 - solar- emissivities
		RS	R1 - solar+ reflectivities
		RS_	R1 - solar- reflectivities
ASF -	Area times gray body radiation exchange factor	ASF	R1 - A script F's
		RNID	R1 - row node id's
		A	R1 - areas
		CNID	R1 - column node id's

Figure 3.9-2: RADSIM/IAC Files